UNIVERSIDAD CARLOS III DE MADRID

Escuela Politécnica Superior

Ingeniería Técnica de Telecomunicación: Sistemas de Telecomunicación

PROYECTO FIN DE CARRERA

Integración de Servicios Web de redes heterogéneas mediante una solución basada en WebSphere DataPower

Autor: Sergio Gómez Pagador

Tutores: Julio Villena Román - María Carmen Fernández Panadero

Octubre, 2015

Título: Integración de Servicios Web de redes heterogéneas mediante una solución basada en WebSphere DataPower

Autor: Sergio Gómez Pagador

Tutor: Julio Villena Román

Co-Director: María Carmen Fernández Panadero

EL TRIBUNAL

Presidente: Ignacio Soto (Ingeniería Telemática)

Vocal: Ana Iglesias (Informática)

Secretario: Jesús Arias (Ingeniería Telemática)

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 15 de Octubre de 2015 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de ___

VOCAL

SECRETARIO

PRESIDENTE

Agradecimientos

A mis compañeros, con los que he pasado más de mil y una aventuras en la Universidad, con los que he sufrido en los malos momentos y con los que sigo sufriendo porque nunca nos podemos ver.

A los profesores que me han ayudado, que me han formado y que, tras tanto tiempo, ya casi no recuerdo.

A los tutores, por guiarme y sobre todo confiar en mí en el momento en que más lo necesitaba.

A mis hermanos, que aunque ellos no lo sepan me ayuda mucho el tenerles ahí.

A Laura, por su apoyo incondicional, por estar siempre a mi lado y por hacer de cobertura cuando más lo necesitaba. ☺

Y a Diego, por animarme con sus juegos y por ser mi ***Sol****.*

Resumen

En la actualidad, y gracias a la red global de Internet, es muy común la comunicación entre aplicaciones web desplegadas en redes locales independientes. Las empresas que proporcionan algún tipo de servicio web, o las que los consumen, se preocupan de mantener una alta seguridad en la comunicación, todo ello además con una velocidad y coste proporcionales.

Así pues, disponer de una capa adicional que proporcione seguridad y que pueda facilitar el tratamiento de las peticiones y respuestas es una ventaja clara para mantener alejados los problemas derivados de comunicaciones inseguras.

La idea fundamental del proyecto es el proporcionar una visión general de una solución a nivel de *hardware* y *software* muy demandada por las empresas para mantener la seguridad en las comunicaciones con el exterior. Dicha solución se introduce como la última aplicación lógica entre los servidores internos y la salida al exterior, siendo el punto de enlace hacia el sistema ajeno.

Aparte de las ventajas en seguridad, se aprovechará también la solución elegida como plataforma para el tratamiento y aceleración de las transacciones que discurren por la misma.

Su descripción y funcionamiento es el tema principal de este proyecto. Mediante una visión general del funcionamiento del aplicativo y de sus características principales, se aplica después al caso concreto de comunicación entre dos empresas ficticias.

Todo ello se apoya en desarrollos realizados a medida y debidamente analizados para poder emplear con exactitud las ideas que se exponen en la práctica.

Índice

1 Introducción y objetivos

1.1 Introducción

En la década de los 90, ante la necesidad de organizar y precisar las interacciones de los nuevos componentes que comenzaron a poblar la Web, surgió la definición de la arquitectura orientada a servicio (o su acrónimo **SOA** – *Service Oriented Architecture*)[1]. Con ella, se creaba un marco de trabajo conceptual en la que una arquitectura u organización fundamental de un sistema debía describirse mediante sus servicios, los cuales deben ser autónomos y granulares, y la relación entre ellos. SOA se presentaba así como una base de trabajo estándar para definir el desarrollo de los nuevos componentes, con el fin no de hacerlos independientes entre sí, sino con la premisa de que puedan comunicarse fácilmente entre ellos.

Dicho estilo de arquitectura define a los servicios como unidades con la mínima funcionalidad implementada. Esto implica que se deban seguir una serie de pautas, como el definir completamente la interfaz, procurar que dependan lo mínimo de otros servicios y que puedan ser reutilizables entre sí. Con ello, se establece un estilo de desarrollo en el que se describe la naturaleza del servicio en sí y no la tecnología aplicada.

En la actualidad, SOA se ha extendido ampliamente y ya no existen prácticamente servicios que no hayan hecho uso de esta definición. Es más, conceptos tales como La Internet de las Cosas (IoT – *Internet of Things*), que busca poder conectar cualquier objeto entre sí a través de Internet, coexiste con SOA y se complementa para converger hacia las nuevas direcciones que toma el mercado[2].

Existen conceptos tales como los servicios web o ***Web Services***[3] (WS), los cuales normalmente van ligados a la definición de SOA. Los servicios web son un conjunto de estándares y protocolos que permiten el intercambio de datos entre aplicaciones. El definir un servicio web no implica que se esté cumpliendo el marco de SOA. Y viceversa, para definir una arquitectura SOA no es necesario ofrecer servicios web.

Como se ha podido deducir, los servicios web van a jugar un papel muy importante dentro del desarrollo de este proyecto: van a permitirnos comunicar las distintas aplicaciones de las redes ya sea dentro de la misma red privada o entre redes heterogéneas. Es por ello que, cuando la comunicación de los servicios web se realice entre redes heterogéneas a través de una red compartida, se dedique especial atención a desarrollar determinados mecanismos de seguridad. Por lo tanto, los principales problemas a solventar van a estar muy relacionados con la seguridad y el rendimiento.

Así pues, el objetivo a seguir a lo largo de este proyecto va a ser el poder implementar diversos métodos de seguridad en una arquitectura SOA genérica, utilizando para ello un aplicativo o máquina que soluciona los problemas derivados de esta arquitectura: **IBM *WebSphere* DataPower**.

---

[1] **SOA** - *Service Oriented Architecture*. Es un concepto de arquitectura de *software* para la cual se diseñan componentes con la finalidad de dar un servicio. Para más información acerca de la arquitectura SOA aplicada al ámbito de los servicios web visitar: [01]

[2] En artículos como el siguiente se discute sobre la aplicación de una estructura SOA para la Internet de las cosas: [02]

[3] Glosario de términos relacionados con los WS: [03]

1.2 Objetivos

La motivación principal para realizar este proyecto es el de analizar el uso de DataPower como solución real para implementar seguridad en las comunicaciones. Para ello se ha partido de una tipología concreta de comunicación: los **servicios web** o *Web Services*.

La comunicación mediante servicios web está muy extendida y normalmente es un foco de problemas en lo que respecta a seguridad y control. De este modo, para aumentar la seguridad en las aplicaciones, para evitar ataques inesperados o para incrementar la capacidad de procesamiento, se opta por utilizar una solución mediante *hardware* y *software* y así solventar esas eventualidades o problemas. Se define como una solución *hardware* porque se utiliza una máquina como *proxy* o intermediario en la comunicación con una alta capacidad de procesamiento y *software* porque dicha máquina utiliza un sistema operativo propio con características avanzadas para el tratamiento de los servicios web. La solución escogida ha sido la máquina de **WebSphere DataPower**[4].

La elección viene condicionada por el buen rendimiento que ofrece, por estar orientada a la seguridad en las comunicaciones, por la potencia y posibilidades de desarrollo y por ser unas de las alternativas más completas dedicadas a estos menesteres. Además es una plataforma con un soporte constante y que evoluciona para ir adaptándose a los nuevos requisitos en cuanto a comunicaciones.

Así que, bajo la situación que se ha presentado hasta ahora, el principal objetivo es el de ofrecer una pequeña guía de iniciación, con ejemplos prácticos, de modo que cualquier técnico especializado en comunicaciones pueda utilizarlo como punto de referencia. Se dispone de mucha información acerca de DataPower en la red, pero ninguna de las referencias consultadas utiliza el enfoque global que se persigue en este proyecto, abarcar desde la descripción genérica hasta los ejemplos prácticos, pruebas, problemas y conclusiones.

Así pues, a modo de resumen, los principales objetivos que se desean alcanzar son:

- Establecer los **conceptos básicos** de las comunicaciones mediante servicios web y **analizar** sus problemas.
- Resumir las **características principales** de DataPower y de sus alternativas.
- Elaborar una **guía técnica** sobre la utilización de DataPower y ofrecer **soluciones** a los problemas planteados.
- Mostrar ejemplos de uso mediante **desarrollos a medida** que simulen la comunicación entre dos empresas.
- Concretar qué líneas de acción se han llevado a cabo correctamente y cuáles pueden ser las **tareas a futuro**.

[4] **DataPower®** es una marca registrada de IBM. La descripción completa del producto, descargas de software, soporte y documentación está disponible en la página web oficial del fabricante: [04]

1.3 Estructura de la memoria

Este proyecto se ha intentado estructurar siguiendo siempre el mismo patrón: se explica desde lo más sencillo y generalizado hasta lo más complejo y detallado. Además, todos los términos que aparecen en la misma se explican a pie de página como nota aclaratoria. Consta de los siguientes apartados:

1. **Introducción**: este apartado que nos ocupa define el tema principal o idea sobre la que se va a desarrollar el proyecto. Se identifica brevemente el contenido que se desarrollará en el resto de apartados.
2. **Estado del arte**: resumen de la funcionalidad de DataPower empezando por la explicación de los servicios web y de su evolución. Se numeran los aplicativos de DataPower en el mercado actual a fecha de este proyecto así como posibles alternativas.
3. **Descripción técnica**: tras la presentación de las arquitecturas comunes en la que se presenta DataPower, se elabora un escenario de utilización y se realiza una descripción detallada de los procedimientos para desarrollar las aplicaciones con DataPower. Todo ello desde un punto de vista meramente técnico, con el que se comienza desde los modelos generales de utilización hasta ejemplos concretos.
4. **Pruebas y resultados**: una vez finalizada la implementación en DataPower de cada una de las propuestas, se realiza una batería de pruebas para cada una con el fin de determinar su correcto funcionamiento.
5. **Conclusiones y líneas de trabajo futuras**: se definen las conclusiones que se han obtenido de la implementación y de las pruebas realizadas en el apartado anterior. A continuación se explican los puntos de trabajo en los que se puede avanzar.
6. **Manual del usuario e instalación**: en este anexo se define la instalación que se ha llevado a cabo para hacer funcionar la máquina de DataPower y cómo se han configurado sus características más importantes.
7. **Referencias y bibliografía**: listado de referencias, manuales y citas digitales o páginas web utilizadas en el proyecto.

2 Estado del arte

2.1 Estado actual y conceptos básicos

Este apartado trata sobre el estado actual de las comunicaciones basadas en servicios web. Para ello es necesario explicar la terminología de los mismos y los organismos que los han ido regulando a lo largo del tiempo para llegar hasta hoy tal y como los conocemos.[5]

Con el fin de estandarizar la comunicación mediante servicios web que estaba proliferando a principios de siglo, existían diversas organizaciones que fomentaban la utilización de modelos de trabajo determinados. Ejemplos de las mismas son **W3C**[6] - "*World Wide Web Consortium"*, **OASIS** - "*Organization for the Advancement of Structured Information Standards*" o la **WC-I**[7] - *Web Services Interoperability Organization*. Dichas organizaciones crearon una agenda de recomendaciones independientemente de cada plataforma, sistema operativo o lenguaje de programación con la finalidad de estandarizar los desarrollos, si bien el W3C abarca un rango más amplio que el de los servicios web.

Las definiciones más importantes en las que se marca el modelo a seguir para trabajar con servicios web son las creadas inicialmente por la WC-I. En la *WS-I Basic Profile Version 1.0 - 1.1* del año 2004 se establece la utilización de los mismos mediante las especificaciones SOAP, WSDL, UDDI, XML *Schema* y HTTPs, las cuales se deben utilizar de manera conjunta con el fin de crear servicios web interoperables entre sí. Otra definición que cobra mucha importancia en el desarrollo de servicios web seguros es la *WS-I Basic Security Profile 1.0*, creada en el año 2007. En ella se establece el uso de protocolos de seguridad para la capa de transporte, seguridad dentro del mensaje SOAP y pequeñas consideraciones para disminuir otros problemas relativos a la seguridad. Es importante señalar que ambas definiciones de trabajo, la *Basic* y la *Basic Security*, se complementan entre sí[8].

Las definiciones de los servicios web de la WS-I se basan en los protocolos o especificaciones de las "Solicitudes de Comentarios" - **RFCs**[9], que se han ido publicando para organismos como la **IETF**, la llamada Fuerza de Trabajo para la Ingeniería de Internet (del inglés *Internet Engineering Task Force*). Este organismo creado en 1986 tiene como principal objetivo crear una red de Internet sencilla y productiva recopilando información técnica de los estándares a utilizar, con grupos dedicados a áreas tales como comunicación, transporte y seguridad. Los objetivos de este organismo se definen en la RFC 3935 proporcionada por ellos mismos[10].

De manera que, según la WS-I, para la definición de los servicios web se propuso la estructura representada en la [figura 2.1], la cual requiere la utilización de diferentes tecnologías descritas por la IETF en sus RFCs, y cada una de ellas aplicada a su correspondiente capa en la pila de protocolos.

---

[5] Para ampliar más información puede consultarse una breve historia de los organismos reguladores en el capítulo 1.6 de la referencia [05]

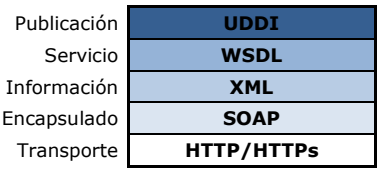
[6] **W3C**: organización dedicada al desarrollo de Web, a la estandarización de protocolos y al fomento de interoperabilidad entre los sitios. Dicha organización ofrece información a través de la web en el siguiente enlace: [06]

[7] **WC-I:** organización con la finalidad de proporcionar una utilización correcta para la comunicación entre los servicios web. A fecha de 2010 continuó trabajando bajo el amparo de OASIS formando la OASIS WC-I: [07]

[8] Para más información acerca de estas definiciones visitar: [08] y [09]

[9] **RFC:** *Resquest For Comments*. Informes técnicos creados para la IETF que definen los estándares para el uso de Internet. Más información en: [10]

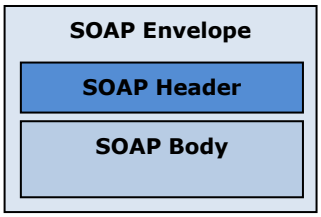
[10] La **IETF** dispone de numerosa información libremente disponible que puede consultarse directamente desde su página Web: [11]



[Figura 2.1] Pila de protocolos para un servicio web

La capa de transporte de los servicios web utiliza normalmente los protocolos **HTTP**[11] y **HTTPs**[12]. Según la definición, los servicios web pueden utilizar ambos protocolos sin problema, cada uno con sus ventajas e inconvenientes. HTTPs implementa, frente a HTTP, seguridad mediante protocolos criptográficos. De este modo, la información que intercambian está cifrada y es menos sensible a ataques externos. El protocolo de cifrado utilizado en el inicio fue **SSL**[13], siendo sustituido posteriormente por sus respectivas revisiones hasta la SSL v3.0 y finalmente por **TLS**[14], mucho más seguro en su definición.

Para la capa de encapsulado del mensaje se utiliza el protocolo **SOAP**[15], que define un mecanismo para la comunicación entre componentes en un entorno distribuido o descentralizado. Ésta se realiza mediante mensajes formateados en XML y transportados por un protocolo de transporte (como se ha comentado antes, no necesariamente HTTP). Una gran ventaja de SOAP es que la comunicación entre cliente y servidor es independiente de la plataforma de cada uno, por lo que las aplicaciones de plataformas heterogéneas pueden comunicarse. Está amparado por la W3C, la cual se encarga de concretar sus características principales.



[Figura 2.2] Estructura de un mensaje SOAP

A continuación se describen brevemente los elementos de la arquitectura de los mensajes SOAP:

[11] **HTTP:** *HyperText Transfer Protocol* o Protocolo de Transferencia de Texto. Protocolo de la capa de transporte de la pila OSI utilizado para la Web. Especifica qué mensajes pueden enviar los clientes a los servidores y qué respuestas obtienen. Cada interacción consiste en una solicitud ASCII, seguida por una respuesta tipo MIME del RFC 822. Todos los clientes y servidores deben obedecer este protocolo. Se define en el RFC 2616: [12]

[12] **HTTPs** es un protocolo similar al HTTP pero utilizando SSL como puente entre la capa de transporte y la de aplicación. En sus inicios se utilizaba sólo SSL, pero se ha ido sustituyendo por TLS para aumentar la seguridad en las comunicaciones al corregir numerosos *bugs*. La definición de HTTP sobre TLS se define en la RFC 2818: [13]

[13] **SSL:** *Secure Sockets Layer.* Protocolo criptográfico que define el estándar para realizar comunicaciones seguras. La versión 3.0 de este protocolo se define en la RFC 6101: [14]

[14] El protocolo **TLS** - *Transport Layer Security* se propuso como una mejora del protocolo SSL v3.0, aumentando la seguridad y solucionando bugs del mismo. Más información en [15]

[15] **SOAP** del inglés *Simple Object Access Protocol*. Definición del W3C en: [16]

- SOAP *Envelope* es un elemento obligatorio que engloba la estructura del mensaje SOAP y define su modo de procesarlo y su codificación.
- SOAP *Header*, que se define dentro del elemento anterior, y que no es más que una cabecera opcional que proporciona información adicional de la transacción. Usualmente contiene información adicional para el procesamiento del mensaje.
- SOAP *Body:* definido dentro del SOAP *Envelope* y que contiene toda la información del mensaje a intercambiar o *Payload*. Por lo tanto, es un elemento obligatorio.

Si bien actualmente se están creando alternativas nuevas, como por ejemplo REST[16], SOAP sigue siendo muy utilizado. Actualmente una gran variedad de sistemas invocan u ofrecen servicios web utilizando para ello el intercambio de mensajes SOAP, y todo ello independientemente de las diferencias entre cliente y servidor que pudieran existir.

Haciendo un breve paréntesis, en lo que respecta a REST, éste no se refiere a una nueva tecnología en sí, sino la definición de un patrón de diseño de arquitectura basada en objetos en vez de en métodos como en SOAP. Principalmente REST mejora a SOAP en los siguientes aspectos:

- Se elimina la rigidez de la definición de la interfaz de la comunicación: deja de existir la relación contractual entre origen y destino que tanto caracteriza a SOAP. Se basa en el protocolo HTTP mediante operaciones definidas: PUT, GET, POST y DELETE.
- Es un concepto más amplio que SOAP, que además no requiere necesariamente la utilización de un formato XML para los mensajes intercambiados.

Para el caso de este proyecto, la elección de SOAP viene condicionada principalmente por la madurez de SOAP frente a REST en términos de estandarización, como ya se demostrado en el apartado anterior, teniendo incluso algunas dedicadas a la implementación de seguridad como por ejemplo *WS-I Basic Security Profile.* Esto propicia la compatibilidad con el aplicativo DataPower del que va a tratar este proyecto, que de manera nativa trabaja internamente con XML.

Como se ha explicado antes, SOAP requiere que la información a intercambiar utilice el lenguaje **XML**[17]. Éste lenguaje se utiliza para describir el contenido Web de manera estructurada utilizando *tags* o etiquetas. Es importante aclarar que no define un formato de datos, sino sólo un contenido: describir el formato de datos es el rol del *XML Schema*[18] y de los DTD o Documentos de Definición de Tipos.

La razón por la que XML sea tan popular como lenguaje de comunicación es obvia. Principalmente, porque permite entender de forma clara y comprensible el contenido del mismo por estar basada en texto y además es operable independientemente de los sistemas que se comuniquen. Otra razón de peso es el que no se necesite un fichero físico para proceder a la comunicación entre pares: en sistemas distribuidos los servicios pueden enviar y recibir flujos completos de datos XML. Asimismo, XML forma parte de la definición de una arquitectura SOA, lo que conlleva que normalmente esté presente en todo tipo de aplicaciones.

---

[16] **REST:** *Representational State Transfer*. Es una definición de arquitectura para una red basada en aplicaciones. Define cómo se identifican sus componentes, cómo interactúan y los protocolos de comunicación. Por lo tanto, puede definirse como un marco de trabajo y no una tecnología. Aplicado al ámbito de los servicios web, implica que es más ligero, tiene resultados entendibles por humanos, *human readable,* y es más sencillo de construir. Para ampliar información se puede consultar la referencia [17]

[17] **XML:** *eXtensible Markup Language*. Lenguaje que permite definir de modo estructurado información codificada para cualquier alfabeto. Más información en el W3C: [18]

[18] **XML *Schema*** proporciona un método para definir la estructura, el contenido y el tipo de dato de los documentos XML. Fue aprobado por las recomendaciones de la W3C en 2001. Más información en: [19]

Además de la información a intercambiar propiamente dicha, también se puede encapsular dentro de los mensajes otro tipo de información relativa con la comunicación. Por ejemplo, dentro de la cabecera SOAP se puede incluir información relativa a las especificaciones de seguridad del mensaje. Mediante estos *tokens*[19] definidos en la cabecera de los mensajes, cliente y servidor son capaces de definir comunicaciones seguras. Del mismo modo, los servidores de seguridad podrían adjuntar información de autenticación, autorizaciones o características adicionales de seguridad sobre los mensajes recibidos, para poder redirigirlos después hacia las aplicaciones de la empresa basada en los *tokens*, lo que se conoce como las *Assertions*[20]. Este tipo de comprobaciones en los servidores abstrae a las aplicaciones de las decisiones de seguridad implementadas.

La comunicación entre aplicaciones que utilizan servicios web no puede definirse completamente si falta el WSDL[21]. Con él se define completamente la utilización de un servicio web, por lo que es requisito indispensable que siempre exista. Gracias a este documento se declaran los requisitos del protocolo y los formatos de los mensajes necesarios para utilizar los servicios que define en su catálogo. Esto es, representa, en formato XML, los datos de los servicios de red así como sus operaciones y puntos de enlace o *EndPoint*.

En último lugar, situado en la capa de publicación de los servicios web, hay que mencionar el catálogo de servicios de negocio: UDDI - *Universal Description, Discovery and Integration.* Está realizado en XML y amparado por la organización OASIS. Este catálogo tiene un concepto similar al de las páginas amarillas y blancas de teléfono, por lo que no es de extrañar que en su definición utilice dichos términos para identificar grupos de servicios web.

Actualmente la utilización de los servicios web requiere la aplicación un amplio rango de conocimientos de distintas tecnologías. No es el objetivo de este proyecto el profundizar en el detalle de todas ellas; además su utilización va a estar limitada al ámbito en el que se desarrolla el proyecto. Aun así, siempre se puede ampliar el conocimiento utilizando los recursos Web proporcionados en los pies de página o en el apartado de bibliografía.

2.2 Desventajas y problemas de XML

Hasta ahora en los apartados anteriores, se ha desarrollado la evolución de las soluciones SOA basadas en servicios web, cómo SOAP tiene un papel fundamental y cómo dichas tecnologías se relacionan entre sí. Se ha explicado además que la utilización de XML en las comunicaciones proporciona muchas ventajas y tiene un apoyo importante de las entidades de estandarización, así que su uso está muy extendido en la Web. Sin embargo, en el otro lado de la balanza están los inconvenientes conocidos asociados a dichas arquitecturas y, en especial, los relacionados con el uso de XML como lenguaje de comunicación. Podemos resumir los problemas principales en dos:

- **Rendimiento**: las acciones que implican tratamiento de datos XML como el "*parseo*"[22], procesamiento y transformaciones de datos penalizan el rendimiento de la máquina de aplicaciones, generando lentitud y carga.

[19] ***Token*** puede ser el nombre, contraseña, identidad, clave, certificado, grupo, privilegios, etc. Se corresponden con el listado de peticiones de seguridad que un cliente requiere del servidor.
[20] ***Assertion***: se refiere a los *tokens* descritos en la cabecera que inserta y valida un servidor de confianza. Por lo que podemos considerar que este dato es confiable.
[21] **WSDL**: *Web Services Description Language.* Más información en W3C: [20]
[22] **Parseo** proviene de la palabra inglesa "*Parse*" y se refiere a las transformaciones entre protocolos de los datos intercambiados, como por ejemplo, de XML a texto simple (*String*). Dado que es un término muy utilizado en la informática, aún sin estar aceptado por la Real Academia de la Lengua, se va a utilizar en este proyecto para referirnos a dicha transformación de protocolos.

- **Seguridad**: existen muchos problemas relacionados con agujeros de seguridad y ataques basados en XML: ataques recursivos, de ejecución de código malicioso, de encapsulación de XML, etc.

Existen muchos ataques conocidos basados en XML, por ello implementar seguridad ante los ataques conocidos puede ser complejo si no se disponen de unos conocimientos avanzados. A modo de ejemplo se presenta la siguiente situación: un atacante envía, dentro de un parámetro de la petición SOAP, un código malicioso basado en la declaración recursiva de los *tags* de tipo *ENTITY*[23]. Este ataque se conoce como de expansión de identidades[24].

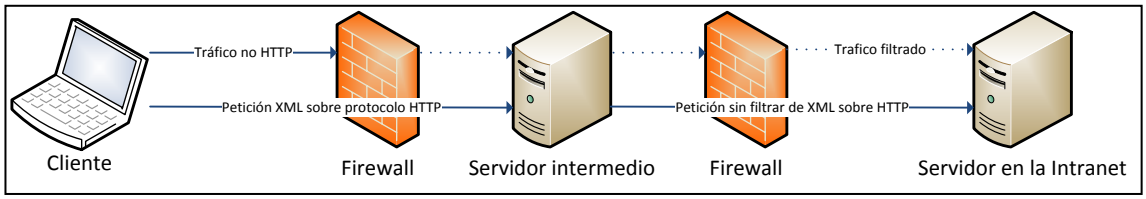
El mensaje malicioso tendría la siguiente estructura:

```
<!DOCTYPE risa [
<!ENTITY ent1 "Ja!">
<!ENTITY ent2 "&ent1; &ent1;">
...
<!ENTITY ent20 "&ent19; &ent19;">
]>
<risa>&ent20</risa>
```

Este tipo de código cargado al inicio del documento XML es capaz de ejecutarse de forma recursiva, penalizando el rendimiento del aplicativo, ya que es capaz de ejecutar cerca de un millón de repeticiones: $2^0 + 2^1 + 2^2 + \ldots + 2^{19}$.

Normalmente trabajar con aplicaciones basadas en XML requiere un compromiso entre seguridad y rendimiento. Es por ello que en muchas ocasiones se recurra a soluciones que disminuyan la seguridad, como no validar el XML, para no penalizar el rendimiento. Esto, unido a la indefensión de las redes tradicionales ante ataques de código XML, provoca que la seguridad de las aplicaciones se considere básica y los sistemas puedan quedar expuestos.

En lo que respecta a la arquitectura, es curioso destacar que una de las principales ventajas de los servicios web, el proporcionar acceso directo al sistema final que nos devuelve la respuesta, sea también uno de sus principales inconvenientes.



[Figura 2.3] Arquitectura de red simple con tráfico de datos XML sobre HTTP

En una red sencilla como la mostrada en la figura, la cual podría corresponder a una pequeña empresa, las peticiones de los servicios web pasan sin filtrar en el *firewall*. El motivo es que el tráfico de datos a través del puerto 80 se utiliza por el protocolo HTTP para navegación Web o para obtener resultados de aplicaciones web dinámicas. El tráfico de este puerto normalmente no es analizado por los servidores para no penalizar el rendimiento de las comunicaciones. Por este motivo, es posible entregar directamente el mensaje XML a la aplicación del servidor de la red interna, pudiendo provocar un error fatal que podría poner en compromiso incluso el funcionamiento del servidor. Existen alternativas o soluciones,

[23] ***ENTITY***: etiqueta XML utilizada para referenciar un objeto en el documento o como acceso directo. Consta de una estructura de tipo nombre-valor.
[24] ***XML Entity Expansion***. Para ampliar información visitar: [21]

como modificar el puerto al que envía la información el servicio web externo. Sin embargo, esto no hace más que enmascarar un problema existente.

Los desarrollos de servicios web son más sensibles a fallos de seguridad. Normalmente el motivo es que, al estar en la zona de confianza o la red local de la empresa, se asuman ciertas garantías de seguridad que, como ya se ha expuesto, no tiene por qué tenerlas. Además, el analizar, validar, "*parsear*" y transformar cada una de las peticiones recibidas por los servicios web requiere una carga de CPU elevada que no suele ser asumible si se trata de un gran volumen de transacciones.

El problema principal de esta situación es que la arquitectura de las redes tradicionales no se diseña para manejar tráfico basado en XML. En el lado opuesto se encuentran las soluciones mediante *software*, que son capaces de trabajar con XML perfectamente, pero no a velocidad adecuada.

Para solucionar estos problemas, se debe utilizar una máquina o aplicativo situado físicamente en una capa entre el exterior y los servicios web de la red local, que permita añadir seguridad y velocidad en el tratamiento de las peticiones. Debe ser capaz de analizar el tráfico XML y realizar los tratamientos de seguridad y transformación que normalmente incrementan la carga de los servidores no dedicados a XML. Esto es, se debe encargar de completar al tratamiento de las comunicaciones XML en las redes tradicionales. Son aplicativos que cumplen los requisitos de una arquitectura SOA y están especializados en el procesado de código XML, proporcionando una solución basada en *hardware* y *software* de alto rendimiento. En este grupo de aplicativos se encuentra el comercializado por IBM: **DataPower.**

2.3 Nociones de DataPower

A continuación se va a explicar en qué consiste DataPower y sus principales características. Para ello se van a desarrollar los siguientes temas:

- **Descripción** de las características principales del aplicativo DataPower como solución a los problemas presentados.
- **Productos comerciales** de DataPower y sus características principales.

2.3.1 Qué es DataPower

DataPower es un aplicativo SOA comercializado por la empresa IBM que consta de componentes *hardware* y *software* a medida para el procesado de XML y que es capaz de proporcionar seguridad adicional en las comunicaciones. Está actualmente bajo el amparo de IBM desde que compró en el 2005 la empresa *DataPower*, que actualmente da nombre al aplicativo comercializado, y que se dedicaba ya entonces a fabricar este tipo de dispositivos. A partir de ese momento, IBM ha estado continuamente evolucionando las características que ofrecía con los productos de esta gama, pero siempre manteniendo el mismo concepto de "todo en uno" que los define.

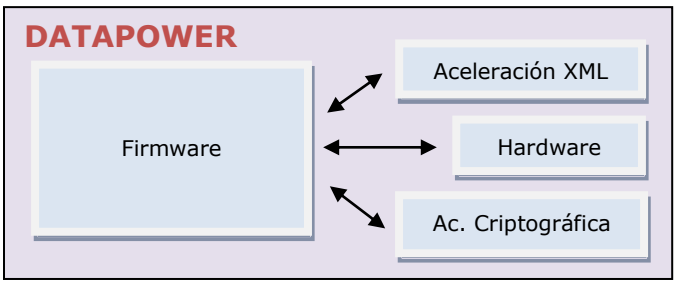
DataPower ha sido definido para sustituir totalmente a soluciones basadas en servidores con *software* dedicado al procesado de XML, con la ventaja de estar fabricado exclusivamente para tal fin, sin un sistema operativo genérico y con componentes *software* a medida. Otra de las ventajas que se ofrece es la seguridad del propio *hardware*: contiene módulos que controlan la apertura no controlada de la máquina para evitar la extracción de

información confidencial (como pueden ser los certificados y firmas de las empresas). Para ello la medida de contingencia que se adopta una vez ha sido abierto es dejar el aplicativo inservible. Como ventaja adicional respecto a un servidor tradicional es el no contar con ningún tipo de puerto o conexión USB para extraer datos o cargar código malicioso. Por lo tanto, se puede considerar a DataPower como una máquina pensada para mantener la seguridad y confidencialidad de los datos que almacena.

En lo que respecta al *software*, para DataPower recibe la nomenclatura de *firmware*[25]. Mediante el mismo se controlan los recursos del dispositivo; además contiene el sistema operativo a medida, por lo que únicamente puede ser utilizado para ejecutarse sobre una máquina DataPower. Este *firmware* está cifrado y firmado, por lo que DataPower no arrancará si el mismo es modificado: tener un sistema operativo cerrado implica que sea más difícil de encontrar vulnerabilidades en el mismo. Además, si fuera necesario, las actualizaciones no son un problema puesto que el *firmware* se puede actualizar fácilmente según se van liberando nuevas versiones por IBM.

El *firmware* controla los recursos del aplicativo e interactúa con los distintos módulos independientes que contiene DataPower:

- Módulo de aceleración de código XML: gestiona el tratamiento de XML.
- Módulo de aceleración criptográfico: tratamiento de todo lo relacionado con criptografía y seguridad tales como certificados, firmas, cifrado, etc.
- Los recursos de *hardware* propios del aplicativo.



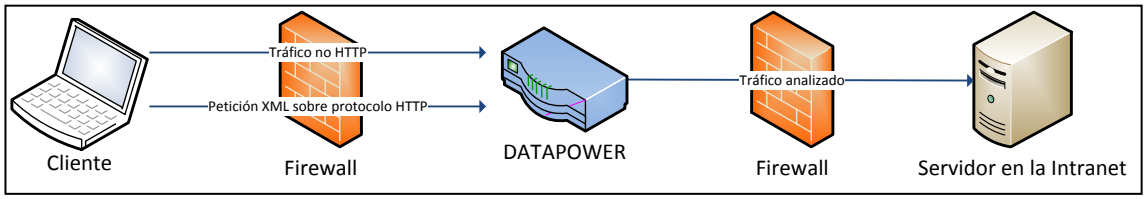
[Figura 2.4] Composición de DataPower

De la figura anterior se obtiene una idea clara: la sencillez del modelo propuesto para las máquinas DataPower contrarresta con el modelo común de los servidores estándar. En éstos últimos el sistema operativo controla los recursos de hardware, los discos duros y las conexiones como CD-ROM y USB. En lo que respecta a las capas superiores de aplicación, existe multitud de aplicaciones de terceros, cada una de ellas con su propia configuración, que proporciona una característica concreta al servidor. Como ejemplo se puede nombrar el *software* siguiente: un servidor web, un servidor de aplicaciones, una base de datos, librerías de XML, librerías de C, librerías de Java, entornos de desarrollo (IDE), etc. Cada una contiene su propia versión de instalación y en consecuencia sus incompatibilidades con otros componentes o versiones. Comparado con DataPower, es mucho más tediosa la puesta en marcha del servidor así como la configuración y actualización de cada componente. Una vez montado el servidor, el rendimiento no es comparable al de DataPower, que tiene una arquitectura cerrada con un *firmware* que aprovecha cada recurso al máximo por estar escrito a bajo nivel. Incluso el nivel de seguridad es menor ya que cada componente

[25] ***Firmware***: es una clase de *software* que controla los recursos básicos del *hardware* disponible, realizado a bajo nivel y específico para el dispositivo en concreto. Un ejemplo de ello son las cámaras fotográficas, que contienen un *firmware* realizado a medida para controlar su *hardware* y hacer funcionar sus componentes. Además puede ser actualizado para corregir o mejorar prestaciones.

depende de sus propias librerías mantenidas por terceros. DataPower es mantenido y respaldado por IBM.

La finalidad de DataPower en su concepción era el de proporcionar una ventaja sobre el tratamiento de comunicaciones basadas en servicios web y XML. En sus sucesivas revisiones tanto de *hardware* como de *firmware* se han ido añadiendo más funcionalidades para adaptarse a la evolución de las comunicaciones entre sistemas. Tal es el caso que en la actualidad DataPower se ha convertido en un aplicativo que da soporte a toda clase de comunicaciones entre redes, soportando cada vez un mayor número de protocolos, y pudiendo incluso actuar como una plataforma integradora de servicios de manera similar a un *middleware*[26]. La gran ventaja de ello es el unificar toda clase de comunicaciones con y desde el exterior, independientemente de la plataforma de cada una, con el consiguiente ahorro de costes.



[Figura 2.5] Arquitectura genérica de red con DataPower

Es por ello que DataPower normalmente se sitúe en la última capa lógica de la Intranet, entre la salida al exterior terminada por el *Firewall* y los servidores internos, de manera análoga a un *gateway*. Como se muestra en la [figura 2.5], sustituiría al servidor intermedio pero con la salvedad de ser capaz de filtrar y analizar el tráfico de datos XML en ambos sentidos. Con ello se proporciona una alta seguridad en los servicios web de la red interna frente a ataques basados en código XML junto con un alto rendimiento, no penalizando las transacciones entre aplicaciones por este tratamiento.

Con DataPower es posible realizar un mayor número de acciones para el buen mantenimiento de las redes basadas en intercambio de mensajes XML. Como ejemplo, se nombran las siguientes:

- Procesar y transformar mensajes utilizando diversos lenguajes basados en XML.
- Controlar los niveles de servicio de los WS implementados en el aplicativo, lo que normalmente se llama en DataPower SLM – *Service Level Monitoring*.
- Cifrado, firma, validación y filtrado de mensajes.
- Enrutar a distintos destinos según el contenido del mensaje.
- Distribuir la carga entre los aplicativos destino.
- Autenticación y autorización de los mensajes – AAA (*Authenticate And Authorize*).
- Cachea mensajes de aplicación.

La mayoría de las acciones de DataPower mencionadas en la lista anterior pertenecen a la capa de aplicación de una pila OSI tradicional. Este hecho contrarresta con el rango de acción de los *proxys/gateways,* los cuales concentran sus características principales en las capas inferiores (de transporte y de red). Es importante destacar este dato ya que demuestra que DataPower no es un mero *proxy* y es capaz de añadir lógica de aplicación adicional.

[26] ***Middleware***: *software* mediante el cual se conectan, gestionan y orquestan las distintas aplicaciones de una red. Un ejemplo de middleware con características de una arquitectura SOA puede verse en la siguiente referencia, desarrollado por Oracle: [22]

2.3.2 Tipos de productos DataPower

A continuación se listan los principales aplicativos que comercializa IBM de la familia de DataPower y sus características principales.

Tipos de IBM DataPower	Características principales
WebSphere DataPower Service Gateway XG45 (virtual y físico)	Seguridad en las comunicaciones: AAA, defensa ante ataques XML, etc.
	Monitorización del Nivel de Servicio - SLM
	Distribución de carga y enrutamiento dinámico
	Integración de sistemas mediante un módulo adicional
	Cuerpo del aplicativo delgado: 1 bahía en el Rack.
WebSphere DataPower Integration Appliance XI52 (virtual y físico)	Características de XG45 +
	Integración de sistemas con funcionalidad total: también con tráfico no XML
	Cuerpo del aplicativo de tamaño de 2 bahías en el Rack.
WebSphere DataPower Integration Blade XI50B/ XI50z	Similar al XI52 pero con otro formato de Rack para su instalación en esqueletos IBM System z.
WebSPhere DataPower B2B Appliance XB62	Características de XI52 +
	Soporte de mensajes B2B
	Soporte a documentos B2B
	Visor de transacciones B2B con capacidad para la simulación de respuestas

[Tabla 2.1] Productos DataPower de IBM

En la [tabla 2.1], el **XG45** se presenta como la solución más sencilla, funcionando como un *gateway* que implementa seguridad a los servicios SOA y con posibilidad de realizar alguna pequeña transformación de código básica. El modelo **XI52** implementa toda la funcionalidad del XG45 y le añade un soporte completo para realizar transformaciones. Debido a este motivo y a que es la elección más popular en el mundo empresarial, se ha escogido este aplicativo para la realización del proyecto. A continuación se nombra el **XI50B**, el cual es practicamente similar en funcionalidad al XI52 pero presenta otro formato de caja, adaptable a otro tipo Rack del centro de procesado de datos en el que se aloja. Por último se presenta la solución **XB62**, que añade soporte para redes basadas en comunicaciones *Business To Business* - **B2B**[27].

Los modelos XG45 y XI52 también se ofrecen en formato virtualizado, lo que significa que es posible ejecutarlo como una máquina virtual en un servidor genérico. Este formato tiene sus pros y contras: ofrece mayor versatilidad frente a un rendimiento y seguridad menor, puesto que se pierde la seguridad contra aperturas y se depende de la capacidad del servidor sobre el que se instale la imagen. Sin embargo, es posible migrar la imagen a un DataPower físico y así evitar esos inconvenientes. De cara a la realización del proyecto se escogió la máquina de DataPower virtualizada por imposibilidad de disponer de la solución *hardware.*

[27] **B2B**. Arquitectura dedicada al comercio de negocios o *e-business* en el que las aplicaciones interactúan entre sí mediante Internet o la Intranet: transacciones financieras, actividades de logística, consulta de stock en almacén, etc. Para más detalles acerca de las comunicaciones B2B y de su relación con el aplicativo de DataPower XB60, al que sustituye el más moderno XB62 visitar: [23]

2.4 Alternativas y conclusiones

En este apartado se van a nombrar las alternativas que existen en el mercado a la utilización de DataPower. Si bien como alternativa siempre se puede utilizar un servidor genérico con una aplicación *Middleware* para sustituir las funcionalidades de DataPower, no es un sustituto real: no es una solución realizada a medida, es una solución heterogénea, y no comparte la finalidad de DataPower (seguridad y rendimiento). Por lo tanto, se descarta el uso de servidores no dedicados a esta necesidad, que además se alejan de la filosofía de trabajo que se está inculcando en este proyecto y que, como se vio antes, no son efectivos para este tipo de tareas. Para finalizar este apartado se aportarán los motivos de elección de DataPower frente al resto de soluciones comerciales.

Se han identificado las siguientes alternativas que trabajan a modo de aplicativo completo y que dan solución a los problemas de seguridad y rendimiento de las comunicaciones SOA entre servicios web:

- Forum Sentry[28]
- CA API Gateway[29]
- Axway API Gateway[30]

No existen estudios ni información concluyente acerca de la cuota de mercado en la que se muestren los datos enfrentados de cada solución. El listado de alternativas, así como la definición de las mismas, se ha obtenido de referencias de los distintos fabricantes y de los estudios de capacidad de la consultora *Lustratus Research*[31].

Como dato curioso mencionar que a fecha de 2007 ya se nombraban cifras de un 56% de cuota mercado para las soluciones SOA de IBM, entre las que se encuentran las soluciones DataPower[32]. Este dato, dada la antigüedad del mismo, no es relevante para determinar quién domina el mercado actual.

Aun así, la balanza parece decantarse hacia el lado de DataPower por motivos de peso: tener su propio entorno de desarrollo embebido para la generación de políticas, programación con código de transformación de XML utilizando plantillas y funciones propias especializadas, disponer de validador de XML interno e incluso tener la posibilidad de programar utilizando el entorno de desarrollo Eclipse. Otro factor a tener en cuenta es el soporte de IBM y de la comunidad de usuarios activa que mantiene los foros[33]. La calidad de la solución aportada por IBM frente a la de sus competidores lo coloca como la mejor opción.

2.4.1 Forum Sentry XML Gateway

Es un aplicativo creado por la empresa Forum Systems, fundada en 2001, que proporciona capacidad de crear políticas de tratamiento de XML, tareas de cifrado y descifrado, así como gestión de identidades de acceso. Estas características son similares a las que puede ofrecer un dispositivo DataPower, que en general suele estar por encima en cuanto a número de protocolos soportados, sobre todo en lo que respecta a redes B2B.

---

[28] Página web del producto: [24]
[29] Página web del producto: [25]
[30] Página web del producto: [26]
[31] **Lustratus Research** es una consultora y analista de las distintas infraestructuras de mercado. Para más información ver: [27]
[32] Ver noticia: [28]
[33] Foro IBM dedicado a la tecnología DataPower: [29]

Productos	Características principales
Forum Sentry XML Gateway	Control de acceso
	Protección frente a ataques XML
	Gestión de identidades de usuarios en las redes
	Gestión de la autenticación, autorización y privacidad
	Funcionamiento como XML Gateway, Mobile Gateway, SOA Gateway, FTP Gateway e Identity Gateway
Forum Sentry XML Gateway con módulo de aceleración hardware ASIC	XML Gateway +
	Módulo HW de aceleración para el cifrado y descifrado de mensajes

[Tabla 2.2] Productos de Forum Sentry

En realidad, las distintas ofertas que ofrece esta empresa corresponden al mismo producto pero ofreciendo un módulo adicional de aceleración hardware dedicado a tareas de criptografía como extra. También ofrece soluciones *software* para proporcionar funcionalidades de manejo de XML como un *gateway* para sistemas operativos Linux y Windows.

2.4.2 CA API Gateway

Esta empresa comercializa un producto derivado del creado anteriormente por Layer 7. Por lo que heredó las características de los productos de una empresa que fue fundada en 2003 para competir en el ámbito de los *gateways* o *proxys* XML, lo que conlleva que disponga de una solución muy completa. El primer aplicativo comercializado se trataba de un XML *Firewall*, el cual ha ido transformandose hasta la actualidad para ofrecer la gama de productos de la CA API Gateway, evolución directa de los de Layer 7.

Productos	Características principales
API Proxy	Ofrece APIs o interfaces de programación especializadas en seguridad y orquestación básica
XML Firewall	API Proxy +
	Protección frente a ataques XML
	Envío basado en el contenido
	Control de la capacidad del tráfico de envío
	Seguridad y gestión de identidades
SOA Gateway	XML Firewall +
	Control de política de tratamiento de transacciones
	Gestión de WS seguros
	Características básicas de funcionamiento como una arquitectura ESB
CA Mobile API	SOA Gateway +
	Acceso unificado o SSO - Single Sign On adaptado a tecnologías móviles (OAuth)
	Mantenimiento y reporte basado en la tecnología SaaS
	Visor de transacciones B2B con capacidad para la simulación de respuestas
	Soporte a protocolos JSON/REST/XML/SOAP
	Cifrado de datos basados en SSL
	Control avanzado del acceso basado en terminales, usuarios o aplicaciones.

[Tabla 2.3] Productos de CA Technologies

La mayoría de los productos comerciales ofrecidos contienen prácticamente la misma funcionalidad y se diferencian entre ellos mediante el uso de ciertas claves que añaden distintos niveles de herramientas. Esto proporciona una ventaja: el cambiar de un aplicativo a otro se puede hacer de manera inmediata insertando las claves correspondientes. Por lo

que la escalabilidad de la arquitectura está garantizada. Todos los productos comercializados se ofrecen en modo físico o virtualizados, tal y como ocurre con los de IBM DataPower.

2.4.3 Axway API Gateway

Este tipo de soluciones se adoptaron de la empresa Vordel, la cual se creó en Irlanda y comenzó como proveedor de seguridad para entornos basados en XML allá por el año 2000. El aplicativo más sencillo, el XML *Firewall*, evolucionó en el SOA *Gateway* y, posteriormente, en el API Server que se comercializa en la actualidad y que fue adquirido por Axway cuando compró la empresa Vordel en el 2012.

Productos	Características principales
Axway API Gateway	Control de acceso
	Protección frente a ataques XML
	Gestión de identidades de usuarios en las redes
	Gestión de la autenticación, autorización y privacidad
	Seguridad y gestión de identidades
	Control del tráfico de datos y gestión de las políticas de seguridad
	Control completo sobre el ciclo de vida de las aplicaciones
Axway API Portal	Capaz de catalogar los WS del mismo y añadir documentación
	Gestión de medidas y de herramientas de auditoría
	Análisis de uso y rendimiento de los WS

[Tabla 2.4] Productos API de Axway

Mediante este aplicativo se ofrecen métodos de seguridad a nivel de mensaje como control y escaneo de cada uno, cabeceras, adjuntos e incluso antivirus. También existe la posibilidad de controlar y rechazar mensajes si el cliente reenvía con mucha frecuencia o si no cumple el acuerdo de interfaz. Estas funcionalidades se complementan con control de identidades, autenticación, cifrado, firma digital, etc.

No obstante, las capacidades de las soluciones como ESB o *Enterprise Service Bus*, característica de la que se hablará más adelante en apartados sucesivos, son bastante limitadas en comparación con DataPower.

Este grupo de soluciones se ofrece en formato *hardware* ofreciendo además disco de almacenamiento RAID o de alta disponibilidad y doble fuente de alimentación. En esta solución se incluye también un módulo para acelerar el código XML.

Pero la característica más importante que ofrece es el entorno de desarrollo que ofrece, basada en Eclipse y de tipo "*drag and drop*" o "arrastra y suelta". Mediante el mismo se pueden definir políticas de tratamiento a los mensajes, incluso tener varias dependiendo del origen. También existen políticas predefinidas para los casos más comunes. Estas características proporcionan un valor añadido muy interesante.

Esta compañía proporciona otros métodos alternativos para la utilización de sus API: su uso de forma conjunta con DataPower. Así pues, se utiliza la API en entornos con máquinas DataPower para paliar las mayores puntos negativos que ofrece el mismo, tales

como el tratamiento de órdenes no XML (JSON y REST). Esto permite complementar las ventajas y desventajas de cada solución para crear un entorno más potente[34].

2.4.4 Conclusiones y elección final

A lo largo de este capítulo se ha descrito el escenario actual y los productos que se pueden utilizar para solucionar los problemas presentados. Se ha realizado un breve estudio de cada uno de ellos presentando las características más básicas a modo de breve presentación o toma de contacto. Sin embargo, el que todos los productos no sean soluciones homogéneas y puedan incluso complementarse entre sí, conlleva que la elección de DataPower sobre el resto sea un cúmulo de decisiones motivada principalmente por la diponibilidad del aplicativo.

Puesto que DataPower no es una solución gratuita y tampoco tiene posibilidad de descarga de una versión de prueba, para el desarrollo del proyecto se ha utilizado el software facilitado por IBM a empresas hermanadas o *partners*. Este software se puede utilizar siempre y cuando sea para fines personales y no lucrativos.

El permitir la disponibilidad del aplicativo en las condiciones anteriores es una estrategia comercial arriesgada, pero que permite estudiar en qué consiste la solución comercial de IBM. Esto unido a sus principales ventajas, tales como su buen rendimiento o su polivalencia, ha guiado la elección hacia el estudio de las posibilidades de este aplicativo frente a sus competidores.

DataPower es un producto en el que IBM invierte gran parte de su esfuerzo para que actualizar sus características, proporcionando la compatibilidad con los últimos estándares, y ofrecer completas referencias o documentación sobre su funcionamiento. Por lo tanto, DataPower se presenta como la opción más completa, polivalente y profesional de las analizadas.

[34] La información del producto ofertado por Axway en un entorno de máquinas DataPower se describe en el siguiente enlace: [30]

3 Descripción técnica

3.1 Estructura del capítulo

A continuación se va a presentar el desarrollo técnico para la realización de una arquitectura SOA con el fin de comunicar dos redes locales a través de Internet; todo ello utilizando una solución basada en una máquina ***WebSphere DataPower Integration Appliance XI52***.

Estructura de la descripción técnica:

1. Presentación teórica de las **arquitecturas lógicas** en la que normalmente se utiliza DataPower.
2. Posteriormente se definen los **objetivos** y la **definición del problema,** donde se exponen las metas a alcanzar y las arquitecturas concretas propuestas con una máquina DataPower.
3. Para cada una de las propuestas anteriores se describe la solución adoptada mediante la implementación de un ejemplo práctico para cada una, ofreciendo así una descripción del funcionamiento a bajo nivel. Consta de los siguientes apartados:
    a. **Arquitectura** de la aplicación.
    b. **Requisitos** generales.
    c. **Diseño** de la matriz de requisitos, que consta de dos partes:
        i. **Aplicaciones de apoyo:** explicación del diseño de las aplicaciones no relacionadas directamente con DataPower.
        ii. **DataPower:** se comenta el diseño llevado a cabo en DataPower para cumplir los requisitos mediante diagramas de flujo.
    d. **Implementación**: descripción técnica detallada de las acciones realizadas en DataPower apoyadas con ejemplos.

3.2 DataPower: arquitectura general de uso

Normalmente DataPower se presenta en entornos reales actuando como *proxy* o servidor de enlace entre dos puntos. En las comunicaciones en las que actúa es capaz de proporcionar características de valor añadido: aceleración de transformación de código, filtrado de credenciales, seguridad, monitorización, etc. Dichos extras son proporcionados por DataPower dependiendo de los requisitos de los proyectos a implementar, no siendo unos requisitos que siempre se apliquen de manera automática.

Por ello se han identificado tres posibles escenarios genéricos de actuación:

- *Web Services Gateway*
- *Enterprise Security Gateway*
- *Enterprise Service Bus*

Para cada uno, DataPower resuelve una necesidad específica del servicio, siendo normalmente problemas endémicos de arquitectura de red relacionados con la seguridad.

Es muy importante tener en cuenta que los escenarios abarcan estructuras generalizadas, cada una con sus propias circunstancias de implementación, por lo que no debe entenderse como una aplicación de las mismas de manera cerrada o estricta; éstas además pueden mezclarse entre sí dando lugar a una definición de DataPower más completa, siempre adaptable a las necesidades que se tengan en cada momento.

Por último, mencionar que los escenarios de uso se pueden replicar a la vez sobre el mismo DataPower, por lo tanto se puede reutilizar para solucionar distintos problemas de la red replicando distintos comportamientos mediante configuración y *software.* Todo ello gracias a la independencia de los desarrollos que en él pueden instalarse.

3.2.1 Web Services Gateway

Actuando como una capa de abstracción para la utilización de servicios web, situándose entre origen y destino, y tradicionalmente sin implementar filtrado de paquetes ni estructuras de seguridad (*firewall*). DataPower se sitúa entre el origen y el destino pudiendo implementar una conexión segura, monitorización de las comunicaciones e incluso pequeñas transformaciones de datos. Es el escenario más habitual en el que se presenta.

Un caso de uso para este tipo de arquitectura es el de la gestión de un servicio web mediante DataPower. Un cliente de la red local se conecta a un servidor externo mediante un WS a través de DataPower, implementando conexión segura. DataPower va a ser el encargado de medir los requisitos y de tratar las peticiones según un acuerdo común entre ambos. Si se cumple, se envía al destino. Implementa seguridad, tratamiento de posibles limitaciones a nivel de servicio, tratamiento de cabeceras a nivel de protocolo de transporte, así como un tratamiento básico de las transacciones realizando transformaciones de datos, etc. Todo ello sin afectar al rendimiento de la aplicación.

3.2.2 Enterprise Security Gateway

DataPower se puede categorizar como un ***proxy*** que añade capacidades adicionales, principalmente relacionadas con las credenciales, a las comunicaciones entre sistemas; esto es, orientado a proporcionar seguridad adicional mediante tratamiento de cifrado, firmado, cabeceras de seguridad, etc.

Como ejemplo se puede citar el caso de utilizar DataPower como gestor de identidades dentro de una Intranet. El usuario o cliente, realizando un único acceso a nuestra aplicación de DataPower, y después de validar la autenticación correctamente en el servidor utilizando la máquina de DataPower mediante LDAP[35] o similares, puede acceder sin autenticarse en el resto de aplicaciones de la Intranet. DataPower utiliza así una cookie LTPA[36] que contiene información de las credenciales y que se inserta en las peticiones hacia los sistemas internos, indicando que es un usuario autorizado. Por lo tanto, una vez dado de alta con un usuario y contraseña, el acceso al resto está garantizado a nivel de sesión. Esto se conoce como *Single Sign-On* (SSO).

[35] **LDAP**: *Lightweight Directory Access Protocol.* Es un protocolo basado en una estructura en árbol utilizado para gestión de directorios activos y totalmente integrado con el protocolo TCP/IP. Este protocolo se define en la RFC 4510: [31]

[36] **LTPA**: *Lightweight Third-Party Authentication.* Es un procedimiento de autenticación de credenciales creado por IBM que consiste en insertar un *token* en la cabecera con información de la autenticación previa para poder reutilizar esos datos en el resto de servidores. [32]

3.2.3 Enterprise Service Bus - ESB

Funcionando como **ESB[37],** DataPower gestiona la lógica de negocio, integrando y gestionando distintos servicios, liberando así de la carga no funcional a los servidores de aplicaciones: seguridad, enrutamiento y tratamiento de peticiones y respuestas. Se puede afirmar que se comporta como un *Middleware*.

DataPower puede realizar transformación de protocolos: envía y recibe conexiones utilizando distintos protocolos. Soporta HTTP, HTTPs, *WebSphere* MQ[38], FTP, Tibco EMS[39] y NFS[40].

También es posible "enrutar" las peticiones, modificar el formato del mensaje, etc.; por lo que es capaz de gestionar diversos destinos analizando datos del cuerpo del mensaje o de la cabecera del mismo. Por ejemplo, dependiendo del rango de IP de la máquina llamante en la petición recibida, se puede gestionar su destino a uno u otro sistema. Del mismo modo, también es capaz de gestionar las peticiones para distintos orígenes, como por ejemplo, tener un cliente mediante HTTP y otro mediante HTTPs que accedan al mismo servicio.

Todas estas características son propias de la definición de un ESB y se realizan en DataPower utilizando herramientas internas, como las acciones predefinidas de las políticas de tratamiento o *policy*, o simplemente mediante transformación de código XML (XSLT[41]).

Un ejemplo de esta arquitectura podría ser un servicio web que, dependiendo del sistema llamante (diferenciado por un parámetro de entrada, por la IP de origen, por una URI, etc.) realizara una serie de acciones internas e incluso acciones contra otras aplicaciones externas. Cada aplicación de E/S conectada a DataPower podría utilizar un protocolo distinto para su comunicación, siendo DataPower el nexo de unión entre todos a modo de bus lógico. Esto podría resumirse en la capacidad de DataPower como orquestador aplicando lógica de negocio para llegar a un fin determinado.

3.3 Objetivos y definición del problema

El objetivo buscado es el permitir la comunicación segura entre los sistemas de una Empresa A (A1, A2,…) con los de otra Empresa B (B1, B2,…) y viceversa, utilizando para ello la red "insegura" de Internet. Por lo tanto, se abarcarán los detalles y escollos a salvar para poder comunicarse de manera segura utilizando una máquina de DataPower. Toda la configuración y programación adicional se desgrana con el objetivo final de poder ser reutilizada como base para futuros desarrollos.

En la [figura 3.1] se muestra la arquitectura genérica de una máquina DataPower en una comunicación entre empresas situadas en redes distintas.

---

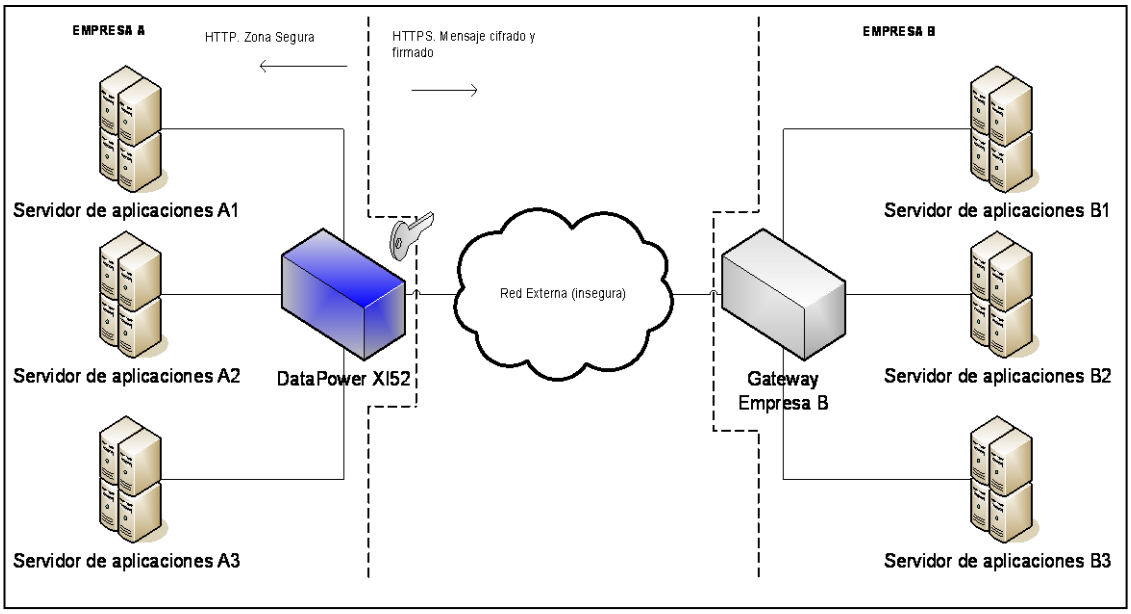
[37] **ESB** o *Enterprise Service Bus* es una arquitectura de software implementada para comunicar mutuamente aplicaciones típicas de una arquitectura SOA. La aplicación ESB se caracteriza por interconectar todas las aplicaciones y de gestionar las relaciones entre ellas.

[38] **WebSphere MQ**: solución de IBM para la comunicación basada en colas. Conexión independiente del sistema origen y destino. [33]

[39] **Tibco EMS**: *Enterprise Message Service* – Solución de Tibco® para la comunicación basada en colas y tópicos. Utiliza un servidor centralizado que gestiona la recepción y el envío de mensajes a suscriptores. Toda la información relativa a los productos de Tibco se encuentra en su página web [34]

[40] **NFS**: *Network File System* – Protocolo de manejo de ficheros en una red mediante arquitectura cliente – servidor. Se describe en la RFC 1813: [35]

[41] **XSLT:** *Extensible Stylesheet Language Transformations*. Lenguaje para transformar elementos XML. Es posible transformarlos en otros elementos distintos de XML tales como HTML, para páginas web. [36]



[Figura 3.1]: Diagrama genérico de arquitectura de comunicación

Teniendo en cuenta las estructuras funcionales descritas anteriormente para DataPower, se han elegido tres escenarios de aplicación, diferenciados lo suficiente entre sí como para poder abarcar las posibilidades reales que ofrece DataPower y, en consecuencia, ofrecer una funcionalidad distinta a las empresas en cada uno de ellos. La elección de este tipo de escenarios viene condicionada por la implantación real que se realiza en entornos productivos con DataPower. El valor de cada uno de los desarrollos radica en la reutilización de acciones entre los mismos. Además, es importante señalar que todas estas estructuras pueden aplicarse sobre la misma máquina de DataPower y funcionar a la vez.

Se han escogido los siguientes roles:

- *Gateway* para comunicación extremo a extremo.
- *Gateway* para gestionar protocolos de transferencia de ficheros.
- *Gateway* funcionando como ESB: implementando una pequeña lógica de negocio.

La gran ventaja de utilizar dichas estructuras es el poder mostrar de manera genérica los ámbitos más comunes de utilización de DataPower. Desarrollando cada uno de los puntos anteriores se ofrece un primer vistazo a las herramientas que nos proporciona esta máquina, lo que finalmente es el objetivo buscado con este documento.

De cara a enmarcarlo en un entorno funcional, se realiza el desarrollo de un ejemplo ficticio para cada uno de ellos. Se simula una relación comercial entre empresas basada en un sistema de compras, gestionando los pedidos y la información de clientes mediante servicios web y definiendo un intercambio seguro de ficheros. En ese entorno, DataPower ofrece seguridad y escalabilidad. Cabe destacar que el desarrollo del ejemplo gira en torno a la implementación de la comunicación desde el punto de vista de la Empresa A.

3.4 Comunicación extremo a extremo

A continuación se describe la arquitectura, diseño e implementación del escenario "*Web Services Gateway*" que se podría utilizar para una comunicación entre dos empresas

simuladas en la que DataPower juega el rol de *Gateway* para una comunicación extremo a extremo.

3.4.1 Arquitectura

En este esquema de implementación se utiliza la máquina de DataPower como puente o *proxy* para la comunicación entre empresas, implementando características tales como seguridad, trazabilidad y control. Para el establecimiento de una comunicación segura se utilizará, en vez del estándar de SSL, el protocolo TLS. Se profundizará más adelante esta elección en el apartado relativo a la descripción de las comunicaciones certificadas.

Se requiere la existencia de **trazas** para la revisión y seguimiento de las transacciones. Por ello, se implementará un sistema de trazas de aplicaciones con guardado en base de datos, así como también utilizando eventos *syslog*[42]. La implementación de este sistema de trazas es una necesidad dadas las limitaciones que tiene DataPower para la gestión de las mismas. Solamente son visibles utilizando la web de administración de DataPower, lo que limita la funcionalidad. Otra alternativa de utilización es utilizar la plataforma de alerta *IBM Tivoli Monitoring*[43]. La elección de la base de datos y de los eventos de *syslog* para cuestiones de trazabilidad, es principalmente, por requerir menos requisitos hardware y por utilizar tecnologías más flexibles, permitiendo que las mismas estén implementadas en un servidor externo dedicado.

Además de lo comentado, se va a realizar la gestión de las peticiones de entrada a nuestra Empresa A según el **SLA**[44] vigente entre las mismas. Estas características se discutirán y justificarán más adelante en la sección de requisitos.

Si bien es cierto que el aplicativo soporta cualquier tipo de comunicación punto a punto, la naturaleza del mismo nos invita a utilizar **XML** como formato del mensaje. Además es un estándar requerido para la comunicación SOAP. Esta elección es beneficiosa puesto que permite la interconexión entre sistemas heterogéneos más fácilmente al definir una estructura estándar en el mensaje para origen y destino.

Otra ventaja de elegir SOAP frente a otros estándares tales como REST o JSON[45], es el soporte que ofrece DataPower con librerías y funciones avanzadas para el tratamiento de las peticiones, seguridad avanzada WS-Security[46] y un motor propio para la aceleración en las transformaciones XSLT.

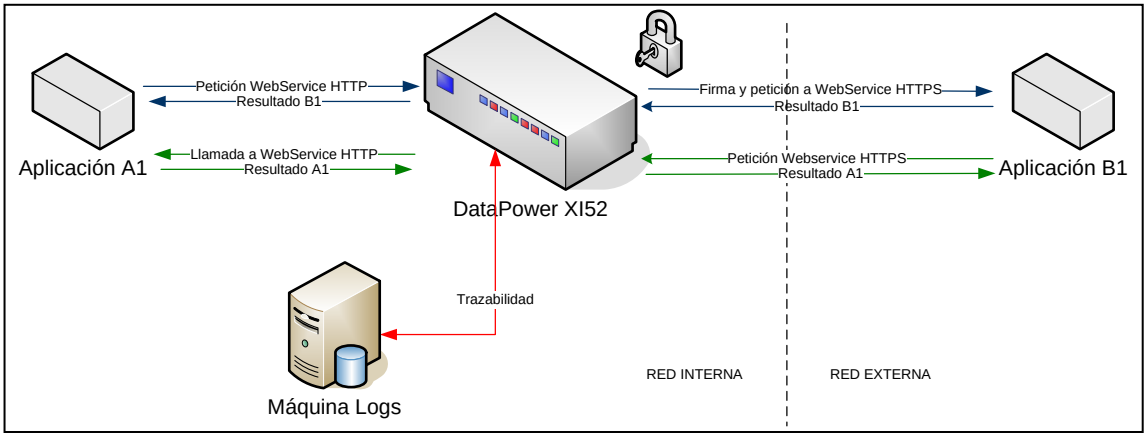
En multitud de foros especializados se comentan las bondades de JSON frente a SOAP y viceversa. No es el objetivo principal de este proyecto el entrar en dicha discusión, sino el definir un medio de comunicación óptimo para una comunicación extremo a extremo con una máquina DataPower. Ésta, de manera nativa, presenta soporte para XML y SOAP frente a alternativas como JSON. Sí es compatible con JSON, sin embargo DataPower utiliza el motor JSONx[47] para transformar la petición JavaScript a XML para después tratarlo de

[42] **Syslog**: envío de mensajes de registro sencillos desde un cliente a un servidor mediante UDP. [37]
[43] **IBM Tivoli Monitoring:** plataforma de monitorización de IBM para sistemas operativos, bases de datos y servidores. Consiste en una solución *hardware* y *software*. Puede obtenerse más información en el *Knowledge Center* de IBM: [38]
[44] **SLA**: *Service Level Agreement* o acuerdo de nivel de servicio es una matriz de requisitos para definir ciertos límites de utilización para los servicios ofrecidos y que normalmente se acepta por ambas partes.
[45] **JSON:** *JavaScript Object Notation.* Mecanismo que define una comunicación basada en *JavaScripts* entre componentes. Se define en la RFC 4627: [39]
[46] **WS-Security**: conjunto de herramientas para aplicaciones de servicios web, basado en estándares de cifrado, criptografía y firmado de XML. Información de la página de OASIS: [40]
[47] **JSONx**: formato estándar creado por IBM para representar JSON en formato XML. Puede obtenerse más información de la página de IBM: [41]

manera más eficiente. Así pues, para poder desarrollar las posibilidades de DataPower independientemente de los protocolos utilizados, se ha optado por el soportado de manera nativa, siendo una solución igualmente válida frente a JSON.

Por lo tanto, se implementará una comunicación SOAP especificada mediante un WSDL. En este ámbito DataPower se comportará como un ***Web Services Gateway***.

En la [figura 3.2] se representa la arquitectura genérica en la que se engloba DataPower. La comunicación consta de dos partes diferenciadas en el sentido: salida desde nuestra red local (mostrado en color azul) y otro de entrada (en verde). Aunque compartan una arquitectura teórica similar, en términos de diseño e implementación en DataPower van a estar claramente diferenciadas, por lo que se hará hincapié en la definición de ambas.



[Figura 3.2]: Esquema de arquitectura de comunicación extremo a extremo

Para la estructura definida en este apartado, se van a detallar dos servicios a modo de ejemplo de implementación, uno para cada sentido de la comunicación. Para el caso del servicio web de entrada se implementará un método para crear un nuevo cliente o usuario de compras dentro de la Empresa A. Para el WS de salida de la Empresa A, se implementará una llamada hacia la Empresa B que proporcione información detallada para un determinado cliente. Esto es, se están creando dos servicios web que utilizan el estándar SOAP para la comunicación entre los sistemas origen y destino, ambos sobre un protocolo de transporte HTTP o HTTPs dependiendo de si el enlace se hace en la red local o desde la externa:

- Servicio web de salida de Empresa A ("*out*"): conexión punto a punto con transformación de HTTP a HTTPs. Ejemplo implementado: recuperar información de cliente externo de la aplicación de compras de la Empresa B.
- Servicio web de entrada a Empresa A ("*in*"): en este caso la transformación es de HTTPs a HTTP. Ejemplo implementado: crear nuevo cliente en plataforma interna de la Empresa A.

3.4.2 Requisitos

Uno de los escollos principales a salvar para el desarrollo de la arquitectura que se muestra en la [figura 3.2] es el relacionado con las **credenciales**. Hay que recordar que todas las comunicaciones que se ofrecen al exterior son de ámbito HTTPs. Ello requiere que exista un intercambio de credenciales para los interlocutores involucrados, por lo que cada

uno de ellos debe poseer un certificado digital válido firmado por una entidad certificadora de confianza (su acrónimo en inglés CA – *Certification Authority*).

Para lograr esta comunicación segura se utiliza el **cifrado de clave asimétrica**[48]: utilizando la pareja formada por el certificado público y la clave privada. Cada empresa debe salvaguardar la clave privada necesaria para descifrar los datos recibidos que se solicitaron con nuestra clave pública.

Para el caso que nos ocupa, dado que es un sistema que no se utilizará en un entorno productivo, resulta inviable realizar una petición de firma de certificado (CSR - *Certificate Signing Request*). Esta petición CSR se genera a partir de una clave o *key* privada que nos identifica como entidad única dentro de un ámbito global. El CSR generado a partir de la misma es enviado para validarse y firmarse por una CA de confianza, como por ejemplo *Camerfirma*[49] o *Thawte*[50].

Pero como se ha comentado, no se va a realizar. Primero, por el coste que ello conlleva, alrededor de 1200 € con 3 años de validez del certificado SSL de *Camerfirma*, o el más asequible de *Thawte,* por aproximadamente 420€ con 3 años; y segundo, porque se tiene que demostrar que la empresa esté proporcionando realmente ese servicio, siendo el *Common Name* – CN definido en nuestro certificado el punto de acceso al mismo o un identificativo de nuestra actividad comercial. Para este proyecto no se está ofreciendo una funcionalidad real al exterior, por lo que no sería admitida la solicitud de firma. Además, con todo esto, la demora podría ser al menos de un mes.

Como solución a la firma de los certificados, se opta por los llamados certificados **auto-firmados** o *self-signed*. Son certificados que representan a una determinada entidad y que están firmados por esa misma entidad. En nuestro caso se crearían dos certificados, uno para la Empresa A y otro para la B, ambos auto-firmados. Para la arquitectura de comunicación presentada es suficiente, ya que sólo entran en juego esas dos empresas. Incluso, siguiendo el razonamiento anterior, no sería necesario realizar el paso de la petición de firma del CSR siempre y cuando se sepa unívocamente qué certificado es el creado por la empresa ajena. Si dicha empresa nos proporcionase el certificado público generado por ellos mismos, podríamos confiar solamente en ese independientemente del firmante. En caso de que una tercera persona crease otro certificado auto-firmado utilizando incluso la misma definición de atributos, no sería válido porque la firma RSA generada a partir de la clave privada no coincidiría.

La generación de los certificados se puede realizar con herramientas como *OpenSSL*, sin embargo, se va a realizar finalmente con las herramientas *Crypto Tools* de DataPower. El motivo de la elección es el poder demostrar las capacidades de creación de certificados y claves mediante DataPower. Además, al realizarlas con el mismo aplicativo, se reducen los trámites para configurar las claves y los certificados en sus respectivos almacenes y se maximiza la compatibilidad por crearse con el formato a utilizar por DataPower, evitando así las transformaciones entre formatos y codificaciones de certificados por estar creados por una aplicación externa. DataPower admite los siguientes formatos para las claves DER[51], PEM[52], PKCS#8 y PKCS#12[53]. En lo que respecta a los certificados, los formatos soportados son los siguientes DER, PEM, PKCS#7[54] y PKCS#12.

[48] La definición de esta comunicación se muestra en el capítulo 8.5 de la referencia [5]
[49] **Camerfirma**: prestador de servicios de certificación al amparo de la LEY 59/2003, de 19 de diciembre, de firma electrónica, es decir como tercero de confianza en las transacciones electrónicas, distribuyendo certificados de identidad que permiten a las empresas identificarse en la red y firmar electrónicamente documentos con total seguridad técnica y jurídica. [42]
[50] **Thawte**: empresa fundada en 1995 que presta servicios como entidad certificadora. En el año 2000 fue adquirida por Symantec. [43]
[51] **DER:** certificado codificado en formato DER (*Distinguished Encoding Rules*)
[52] **PEM:** certificado codificado en BASE64.

En lo que respecta a mantener la **seguridad de la red**, no hay que permitir que los WS presentados en la red local sean accesibles desde fuera de nuestra red de confianza, ya que implicaría un agujero de seguridad importante. El caso más habitual de arquitectura con DataPower en una red privada de empresa es utilizar una única interfaz para unificar las comunicaciones entrantes y salientes, y se deja la gestión de la IP pública y del acceso a la máquina en manos de los administradores de redes. En nuestra configuración no disponemos de dicha infraestructura, así que de cara a blindar los servicios prestados levantados en local sobre un protocolo HTTP, se utilizarán dos interfaces de red: la primera se va a utilizar para levantar los servicios web de acceso privado, mientras que la segunda se utilizará para los servicios web externos. Por lo que desde el exterior no se podría llegar a la dirección sobre la que se montan los servicios HTTP en la IP interna, solamente a los proporcionados por la segunda interfaz con HTTPs.

Una característica muy común ofrecida por DataPower en las estructuras de tipo *Web Services Gateway* es la monitorización de las transacciones. Para ello, el WS de entrada a la Empresa A va a implementar un medidor de **nivel de servicio** o SLA. DataPower permite la posibilidad de realizar este tipo de control de manera nativa, por lo que es una buena idea el implementarla puesto que es una característica reutilizable para todo tipo de desarrollos. Se utiliza para limitar, en cierta manera, el servicio que se ofrece a los sistemas externos de cara a cumplir los distintos contratos con cada una de las empresas y así no saturar el servicio. Como medida de control preventivo es un requisito que sí debe estar dentro del desarrollo de cualquier servicio ofrecido. Para nuestros ejemplos en concreto, este medidor es referido al número de peticiones límite a la hora, al día y de manera concurrente que se pueden ejecutar para eses servicio web. Se ofrecen más alternativas de medición en DataPower, pero se utilizan las comentadas por ser las más utilizadas en el ámbito profesional.

Otro requisito a cubrir definido previamente en la arquitectura es el de poder gestionar las **trazas de ejecución**. Para la gestión de dichos logs de ejecución se opta por utilizar una base de datos sita en un servidor externo. Como se comentó antes, DataPower no ofrece la posibilidad de gestionar logs de manera nativa, o al menos no en el sentido práctico puesto que sólo nos incluye información de errores de sistema, por lo que hay que utilizar otro tipo de soluciones. Para ello, la más lógica es la de utilizar una base de datos para poder recopilar los datos de las peticiones y respuestas, lo cual es muy útil en caso de realizar estadísticas de las mismas a posteriori o para comprobar en qué estado terminó una petición o respuesta anterior. Utilizar una base de datos para guardar la información nos proporciona agilidad, compatibilidad y potencia.

Como se presentó en la arquitectura, existen alternativas a este guardado en base de datos, como por ejemplo la utilización de *IBM Tivoli Monitoring,* que se han descartado por su alto coste de implementación: altos requisitos de software/hardware y el ser una solución comercial cerrada, que incrementaría el coste final del proyecto (tanto por necesitar serviores adicionales al ya existente como por aumentar los recursos para su implementación).

Respecto a la **estructura de datos** a almacenar, debe ser lo suficientemente completa como para tener los campos del XML que identifiquen unívocamente cada petición y respuesta. Se anula la posibilidad de almacenar en base de datos el documento XML completo de la petición debido a que si el volumen de peticiones y respuestas aumenta, el espacio disponible a requerir sería desproporcionado. Esto implicaría además el dimensionar

[53] **PKCS#12:** o comúnmente conocido como formato *.p12*, es un contenedor de certificados en el cual se incluye también la información de la clave privada y del árbol de certificados. La clave privada se protege con una contraseña.
[54] **PKCS#7:** contenedor de certificados con extensión *.p7c* que únicamente contiene información de certificados o de listas de revocación de certificados. Sólo muestra información de la clave pública.

u optimizar la base de datos para poder tratar grandes cantidades de información XML e insertarlo en determinadas tablas.

Así pues, aprovechando la aceleración de código XSL que proporciona DataPower se puede tratar la respuesta obtenida, extraer la información de los campos intercambiados entre origen y destino e insertar sólo los datos que interesen en la base de datos. El impacto de procesamiento de esta elección se antoja bajo, ya que DataPower gestiona realmente bien el código XML. Se realizarán las mediciones pertinentes en el apartado de pruebas para corroborar esta afirmación.

Además de la gestión de **trazas** de peticiones y respuestas utilizando una base de datos, se realizará una exportación de trazas de sistema del aplicativo DataPower a una máquina remota para poder gestionarlos desde un servidor externo. Estos logs o trazas se guardan por defecto en la memoria interna del dispositivo. Por lo tanto, se creó esta condición como requisito para aumentar la calidad de la trazabilidad de DataPower y el valor de la solución. Al tener las trazas en un servidor externo, en caso por ejemplo de que exista un error fatal en DataPower, se puede acceder al mismo para ver el detalle de los logs antes del fallo.

Para crear este requisito, se ha pensado en utilizar la misma máquina que da soporte a la base de datos; así pues, dará soporte además a esta recepción de eventos *syslog*, guardando los **logs de administración** de DataPower en un fichero de texto plano.

La elección de este tipo de comunicación es clara. Por una parte, los servidores basados en Linux utilizan internamente un guardado de logs mediante un demonio de *syslog,* lo que nos permite que al configurarlo adecuadamente pueda recibir eventos externos de cualquier otro sistema. Por otra parte, DataPower soporta este tipo de comunicación para el envío de los logs de administración o sistema a un servidor externo. Puesto que el servidor al que vamos a enviar los logs pertenece a la misma subred de la empresa, con lo cual pertenece a nuestro ámbito de confianza, se puede utilizar la comunicación de eventos *syslog* sin problemas*.*

Recopilando todo lo anterior, estos son los requisitos a la hora de realizar el diseño de los servicios web de una comunicación extremo a extremo:

- **Credenciales** de las comunicaciones: utilizando certificados auto-firmados generados con las *Crypto Tools* de DataPower y firmando las respuestas para que la comunicación hacia redes externas no se vea comprometida.
- **Blindar** físicamente los servicios web HTTP utilizando distintas interfaces de red: habilitamos dos de las conexiones de red de DataPower para levantar los servicios HTTP y HTTPs en cada una.
- **Acuerdo de nivel de servicio**, SLA: se genera un límite de ejecuciones por día y por hora de cara a no saturar el servicio. El resto de peticiones que sobrepasen dicho límite son desechadas.
- Implementación de logs en **base de datos**: utilización de una base de datos Oracle para almacenar las peticiones y los datos relevantes de los XML obtenidos en cada una de las llamadas.
- Implementación de **logs administrativos**: utilizando comunicación con un servidor de apoyo mediante eventos *syslog*.

3.4.3 Diseño

Para el diseño de este tipo de comunicación se va a diferenciar el diseño de las aplicaciones de apoyo y las del propio DataPower.

3.4.3.1 Diseño de las aplicaciones de apoyo

En los apartados que se desarrollan a continuación, se va a mostrar el diseño de los requisitos no relacionados directamente con la tecnología DataPower, es decir, se amplía el detalle acerca de la estructura de red local, de la base de datos para logs funcionales, de la arquitectura específica de la red, de la generación de certificados y de la configuración de eventos de sistema.

a) Red local

Este apartado quiere arrojar un poco de luz sobre ciertos aspectos o consideraciones que se deben tener en cuenta para poder instalar nuestras máquinas en la red local. No es exclusivo de DataPower, sino que involucra a todos los servidores utilizados para nuestro propósito.

Nuestro objetivo es el crear una pequeña **red segura** o de confianza en la que se pueda replicar un entorno de desarrollo con DataPower. Para ello se debe tener en cuenta que seamos administradores de la red a tratar. En nuestro caso se ha implementado en una red local formada por un conjunto de aplicaciones o máquinas de confianza conectadas al *router.* Para los todos los ejemplos ilustrados esta red va a ser la red de la Empresa A.

Para integrar la máquina de DataPower en la red local e implementar seguridad se utilizarán dos interfaces. DataPower ofrece un número mayor de ellos, pero en nuestro caso no va a ser necesario utilizar más. La primera de ellas va a estar ligada a las ejecuciones HTTP (el nombre interno utilizado por DataPower va a ser *eth0*). En concreto, va a estar asociada a las peticiones que realicen las aplicaciones internas de la Empresa A al exterior, por ello se utiliza un protocolo de comunicación no seguro. De cara a las peticiones que llegan a nuestra red segura se levantarán los servicios web con el protocolo HTTPs en la segunda interfaz (*eth1*). Por lo que, los servicios web que reciban peticiones desde el exterior utilizarán un protocolo seguro.

Respecto a la configuración del *Gateway* de salida o puerta de enlace, se realiza filtrado de la dirección MAC[55] para las conexiones salientes de modo que solamente la máquina DataPower XI52 pueda tener acceso al exterior. Por ello, se limitan las conexiones directas desde cualquier aplicativo de la red local para que no acceda al exterior si no es a través de DataPower. Se evita así que desde las aplicaciones se puedan realizar llamadas directamente al destino sin pasar por DataPower y saltándose los protocolos de seguridad al no implementar ni cifrado ni autenticación.

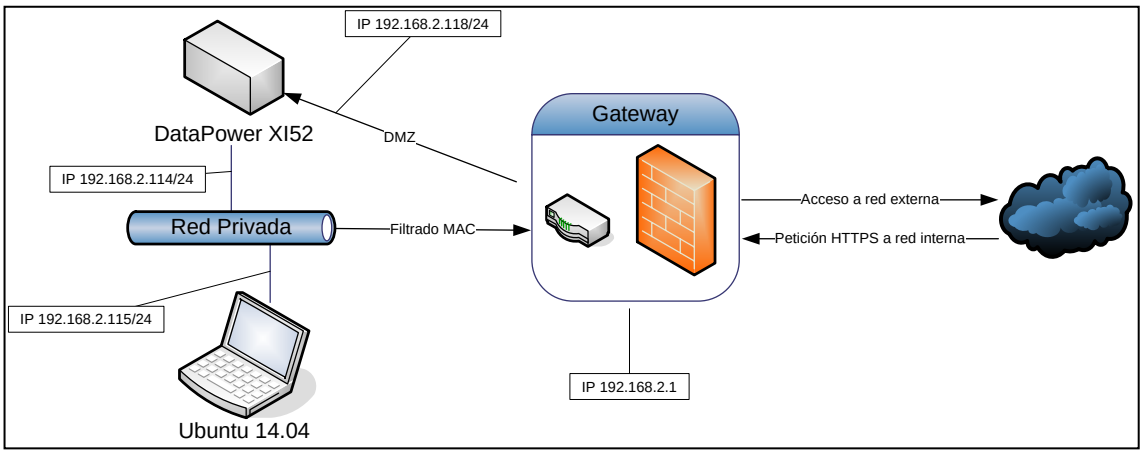
Típicamente, DataPower se configura en la red **DMZ**[56] para proteger a la red interna frente a ataques, gestionar las conexiones SSL y controlar el acceso de clientes de fuera de la red. Es decir, se confía en DataPower la gestión de la seguridad de las comunicaciones entrantes y salientes. De este modo DataPower va a ser el encargado de gestionar la comunicación y el acceso a los distintos servicios ofrecidos por y para nuestra empresa. Esto implica que se deba definir un impacto en DataPower cada vez que se añadan o modifiquen requisitos de las aplicaciones: modificación de los parámetros, añadir nuevas funcionalidades, etc. Por ejemplo, cada vez que alguna nueva aplicación de nuestra empresa

[55] **MAC:** *Media Access Control.* Identificador con una longitud de 48 bits asociada a una tarjeta de red, por lo que normalmente se le conoce como dirección física. Esta dirección es única para cada dispositivo de la red global.
[56] **DMZ**: *Demilitarized Zone* o Zona Desmilitarizada representa una sub-red física o lógica que separa la red local interna de las redes no seguras, como puede ser Internet. Proporciona una capa adicional a la red de acceso e impide a un atacante acceder directamente contra la red interna.

ficticia tenga que acceder a una entidad externa se debe realizar su correspondiente desarrollo en DataPower para permitir la comunicación.

La situación descrita anteriormente se representa en la [figura 3.3]. Sobre la misma también se muestran las IP utilizadas en el caso particular de que la configuración utilizada en nuestras simulaciones, lo cual tiene efectos meramente informativos puesto que depende de la configuración de cada subred. En nuestro caso:

- *eth0*: 192.168.2.114/24, con *Gateway* por defecto igual a 192.168.2.1 (el *Router*).
- *eth1*: 192.168.2.118/24, utilizada para levantar los servicios web HTTPs.



[Figura 3.3]: Diseño de la conexión de los componentes en la red privada

Nuestra red local consta de dos máquinas: una máquina DataPower y un servidor secundario desde el que se realiza la inyección de peticiones *SOAP* y el tratamiento de los logs. Ambas máquinas pertenecen a la misma subred (*255.255.255.0*) y, en nuestro caso particular, son máquinas virtuales. El utilizar máquinas virtuales va a permitir simular la infraestructura desde una única máquina física y utilizarlas como si estuviesen directamente conectadas a la red. Éste ha sido uno de los objetivos buscado: poder disponer de un entorno sencillo en el que ejecutar los distintos servidores implicados sobre una misma máquina utilizando virtualización.

b) Comunicaciones certificadas

Utilizar comunicaciones certificadas es un requisito indispensable cuando se quiere mantener un cierto nivel de seguridad. Los certificados van a representar las **credenciales** de comunicación de los interlocutores activos para una conexión segura. Para realizar una conexión entre pares bajo un protocolo seguro se utilizará el estándar TLS 1.2, para así evitar *bugs* detectados en versiones inferiores o en la anterior denominación del protocolo, SSL (por ejemplo el conocido *bug* llamado *heartbleed*[57]).

También es muy interesante el permitir sólo la comunicación entre determinados interlocutores. Al únicamente definir una serie de interlocutores definidos como válidos, se limita la utilización del servicio para otros no autorizados; muy útil dado que nuestra

[57] **Heartbleed:** fallo de seguridad del protocolo SSL en su versión 1.0.1f, descubierto el 1 de Abril de 2014, que permite a un atacante obtener información sensible del servidor utilizando un sobredimensionamiento del buffer en las peticiones de *heartbeat*, utilizadas para comprobar si el destino está o no disponible.

comunicación va a realizarse a través de una red insegura como Internet. El método para conseguir esto es utilizando la llamada autenticación mutua. Ambos interlocutores, cliente y servidor, se identifican entre sí de modo que se tenga en el servidor una lista de clientes reconocidos como válidos y viceversa, no permitiendo el establecimiento de la comunicación para los no confiables.

Para el caso particular de la comunicación entre las dos empresas, cada una va a tener su certificado independiente. Si bien es cierto que, como se ha comentado antes, van a ser auto-firmados, va a ser suficiente para que podamos identificar unívocamente al interlocutor. Por lo tanto es requisito indispensable que cada empresa deba identificarse con su propio certificado incluyendo además en su lista de certificados de confianza el de la empresa ajena. Al final se busca realizar una identificación mutua de ambas empresas basada en un par de certificados auto-firmados. Además, como característica adicional, se incrementa la seguridad ante alteraciones en el mensaje firmando la respuesta con la clave privada, ya que se puede utilizar el servidor de DataPower para realizarlo.

Para la creación de un certificado, en primer lugar hay que definir la **clave privada**. Para la emisión de la misma se utilizarán las *Crypto Tools* configurando una longitud de codificación de 4096 bits, la más segura y robusta que nos permite DataPower. Se crea además con una duración de 730 días. Si se fuera a crear un certificado temporal, se debería reducir dicha duración. En nuestro caso se opta por la duración típica para un certificado de entornos productivos o reales. Es obligación de cada empresa el custodiar dicha clave, no ofreciéndola a terceros, y el volver a generar un certificado válido cuando éste caduque a partir de la misma.

Country Name (*C*)	SP
State or Province (*ST*)	Madrid
Locality (*L*)	Carabanchel
Organization (*O*)	"Empresa X"
Organizational Unit (*OU*)	DataPower "Empresa X"
Common Name (*CN*)	webservices."Empresa_X".es

[Tabla 3.1]: Datos organizativos de la clave privada

En la definición de la clave privada es necesario precisar el *subject* o suscriptor, que identifica quién es la entidad a la que representa. Dicho suscriptor se compone de diversos campos identificativos como los mostrados en la [tabla 3.1], siendo el único obligatorio el *Common Name* – CN.

Los certificados generados a partir de la clave privada constan de un campo que representa la **entidad certificadora** del mismo, el *Issuer* o emisor. Al ser auto-firmados, el contenido de ese campo va a ser similar al del *subject.* Normalmente este emisor es una entidad certificadora de confianza, como *Camerfirma o Thawte*.

Es necesario que, para evitar problemas, DataPower **confíe** sólo en unos certificados determinados. El modo de realizarlo es importando los certificados públicos de las entidades de confianza a las que permitiremos la conexión; en nuestro ejemplo es precisamente el creado por la empresa ajena, la Empresa B. Por lo tanto, al sólo disponer de dos entidades a comunicar, lo más sencillo y práctico es que cada una confíe en la otra y que los certificados sean auto-firmados.

No se va a profundizar más en este tema ya que la situación de tener certificados auto-firmados es un requisito para poder realizar los ejemplos. Para un sistema real o uno que se utilice en un entorno productivo, se recomienda encarecidamente no utilizar

certificados auto-firmados. Al menos se recomienda que estén firmados por una entidad de confianza en el ámbito de la empresa local, siempre que haya conocimiento de dicha operación por parte de los implicados. La firma de un certificado por una entidad certificadora de confianza es la opción más segura ya que son entidades con un reconocimiento global.

En nuestra empresa, DataPower será el encargado de **almacenar y gestionar** la clave privada y los certificados. Una vez generada la clave o *key* desde el propio dispositivo, sólo se tiene una oportunidad para obtenerla si no se dispone de un módulo *hardware* adicional en DataPower llamado HSM (*Hardware Security Module*). Por ello, a la vez que se crea la *key* en el directorio *cert* por defecto, se puede también exportar al directorio *temporary* si así se indica. Desde dicho directorio se puede descargar sin problemas, y permanecerá con esos datos hasta que el aplicativo se reinicie o se borren los ficheros de modo manual. Si no se realiza su exportación en ese momento, más tarde no estará disponible, pues no se puede extraer información relativa a claves privadas de DataPower por cuestiones de seguridad. Por ese motivo se elige a DataPower como almacén de claves y certificados; la seguridad del mismo es muy elevada, teniendo incluso sensores de apertura del dispositivo.

En caso de tener que volver a crear otro certificado, por ejemplo por **caducidad** del actual, se puede realizar desde DataPower mediante las *Crypto Tools* sin problemas, aún sin conocer la palabra clave o *password* de la *key*. Sólo con volver a utilizar el mismo objeto *key* creado inicialmente en DataPower se puede obtener un certificado auto firmado o una CSR para su firma por una CA.

c) Base de Datos para trazas de ejecución

El diseño de la **base de datos** requiere un conocimiento previo del "modus operandi" de DataPower. Como se argumentó anteriormente, es necesario un sistema externo que apoye a DataPower en la tarea del guardado de trazas de ejecución. Por ello se optaba por realizar dicho apoyo con una base de datos esgrimiendo ventajas como sencillez en la implementación, coste asociado respecto a otras alternativas como IBM *Tívoli* y gestión de los resultados, referido a la facilidad de realizar extracciones de los datos para posibles informes de peticiones ya realizadas.

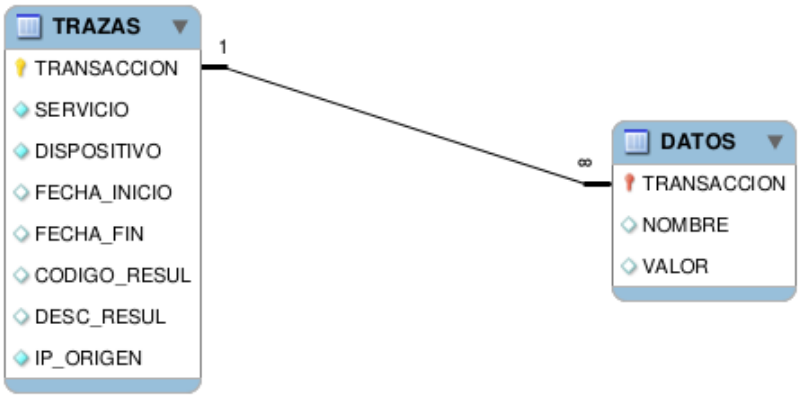
La base de datos se ha diseñado pensando en la sencillez y en la funcionalidad que debe ofrecer como arquitectura para el guardado de trazas. Siempre hay que tener en cuenta que la información a almacenar dependerá de nuestros requisitos individuales de proyecto aunque, como referencia a otras implementaciones, puede utilizarse el esquema propuesto, ya que ofrece información básica de las peticiones y respuestas que cursa DataPower.

En nuestra arquitectura, la base de datos está ejecutándose sobre la máquina virtual Ubuntu 14.04. No cuenta con una gran capacidad de procesamiento, lo cual finalmente no importa para nuestras pruebas. Lógicamente, para entornos productivos sería necesario redimensionar la capacidad de la máquina. En ese caso es muy normal contar con servidores modulares UNIX dedicados en exclusiva para tal tarea. Se escapa del ámbito de este proyecto el estimar una máquina para un sistema productivo real.

Como base de datos se elige la solución proporcionada por **Oracle**. Las bases de datos soportadas por DataPower son Oracle, DB2[58], *Sybase*[59], *Microsoft SQL Server*[60] e

[58] **DB2**: base de datos relacional de IBM que cuenta con un motor específico para el tratamiento de campos XML de forma avanzada (búsquedas e indexación). [44]
[59] **Sybase:** conocido como *SAP Sybase IQ* es una base de datos orientada a ofrecer soluciones de inteligencia empresarial (reportes, análisis de datos, etc.).

IMS[61]. La decisión final es una cuestión de prioridades: se requiere una distribución gratuita, con soporte para máquinas Linux, de fácil configuración y con una amplia comunidad detrás. No es un requisito indispensable el manejar datos en formato XML pues se extraerán sus valores con DataPower. En caso contrario, la elección hubiera sido DB2 de IBM, principalmente por la compatibilidad con DataPower. De todos modos, la estructura que se va a describir se puede configurar independientemente de la base de datos utilizada. Para nuestro propósito, la de Oracle es la que se ajusta al modelo de datos descrito; aunque la elección siempre puede ser una cuestión de afinidad o de simple reutilización de otra base de datos ya existente. Oracle proporciona una solución gratuita con menor funcionalidad llamada Oracle XE[62] y que es perfectamente adaptable a la arquitectura a utilizar. Basaremos los ejemplos prácticos en esa distribución.



[Figura 3.4]: Modelo de datos para las trazas de DataPower

En lo que refiere al diseño de la misma, el *tablespace*[63] o almacenamiento lógico de base de datos se va a componer de dos tablas llamadas TRAZAS y DATOS. La primera de ellas está pensada para tener una única inserción por petición a cada servicio web. La segunda contiene un registro por cada dato XML de la petición SOAP a almacenar, y consta de una estructura del tipo nombre-valor. Los campos a insertar, así como la estructura de la tabla se puede ver en la [Figura 3.4].

Este es el detalle de los campos para la tabla TRAZAS:

- TRANSACCION: identificador único de la transacción. Es un dato de tipo *NUMBER(30)* y nunca puede ser *NULL*. Este número proporcionado por DataPower es único para cada petición, por lo que se utiliza como clave primaria de la tabla (*Primary Key* – PK). La otra opción sería utilizar un incremental para dicho valor que gestione la misma base de datos. Sin embargo, esta situación no es recomendable ya que, como se verá más adelante, el número de transacción se puede utilizar para buscar el detalle de las trazas XML con la utilidad *Probe* de DataPower.
- SERVICIO: nombre de la URL del servicio web ejecutado en DataPower. Es un tipo de dato de texto *VARCHAR2(60)*. El valor de este campo nunca puede ser *NULL*.

[60] **Microsoft SQL Server:** base de datos de la empresa Microsoft sólo disponible para sistemas basados en Windows.
[61] **IMS:** gestor de bases de datos de IBM.
[62] **Oracle XE:** *Oracle DataBase Express Edition.* Edición de base de datos de Oracle gratuita y de libre distribución con soporte para Windows y Linux. [45]
[63] **Tablespace:** es la unidad lógica de almacenamiento en una base de datos Oracle.

- DISPOSITIVO: nombre del aplicativo de DataPower por el que se ejecuta la petición. Es un tipo de dato *VARCHAR2(30)* y nunca puede ser *NULL*. Identifica la máquina física desde o hasta la cual se envía o recibe la petición. En nuestro sistema no es de gran utilidad puesto que sólo se utiliza una única máquina de DataPower.
- FECHA_INICIO y FECHA_FIN: campos de tipo *TIMESTAMP(6)* que almacenan la fecha de inicio y la de fin de cada petición respectivamente. En primer lugar, al realizar la llamada del servicio web, se rellena el campo de fecha de inicio. Cuando se recibe la respuesta, es cuando se rellena el campo de fecha de fin. Por lo tanto, la resta de ambos campos proporciona el tiempo de ejecución del servicio web.
- CODIGO_RESUL: código de resultado de la operación realizada. Es un campo de tipo *VARCHAR2(60)* y puede ser *NULL*. El valor del mismo se rellena con un OK o con un código de error una vez se tenga respuesta de la operación o en el caso de que se ejecutara alguna regla de tratamiento de errores.
- DESC_RESUL: descripción del resultado. Es un tipo de dato *VARCHAR2(256)* y puede ser *NULL*. Como el campo anterior, se rellena en caso de error con su descripción detallada.
- IP_ORIGEN: identifica la máquina llamante mediante su IP. Es un tipo de dato *VARCHAR2(50)* y puede ser *NULL*.

A continuación se muestra la descripción para los campos de la tabla DATOS, consta de una estructura de campo-valor:

- TRANSACCION: identificador único de la transacción. Es un dato de tipo *NUMBER(30)*
- NOMBRE: nombre del campo almacenado en la dupla de valor. Es un tipo de dato *VARCHAR2(256).*
- VALOR: valor del campo identificado con el número de transacción. Es un tipo de dato *VARCHAR2(256).*

Estas dos tablas están relacionas por el parámetro TRANSACCION, siendo una clave foránea[64] con una relación entre las mismas de "1 a n". Por lo que borrar un registro en la tabla primaria TRAZAS, conlleva borrar los registros relacionados en la tabla secundaria DATOS definidos por el parámetro TRANSACCION. Con esta relación queda resuelto el problema de que las filas borradas de la tabla TRAZAS no se borren en la tabla DATOS, ocupando espacio de la base de datos y provocando finalmente fragmentación y degradación de la aplicación.

d) Trazas de sistema de DataPower

Las trazas de sistema están siempre habilitadas en DataPower, sin embargo, el poder acceder a ellas sólo es posible desde la consola de administración y no de un modo demasiado intuitivo. Es muy útil tener los logs de sistema en un servidor externo y no sólo en DataPower, puesto que, por ejemplo, en caso de que se encuentre inoperable y no se pueda acceder por un error, se puede determinar qué ha propiciado el fallo. El diseño mencionado aquí es común para toda implementación de los servicios web realizados bajo el mismo dominio de DataPower y puede generalizarse a cualquier implementación que requiera un sistema externo de logs. Aclarar que las trazas de sistema de DataPower se gestionan a nivel de dominio y cada uno requiere su configuración independiente.

La solución finalmente elegida es realizar la gestión mediante el envío de eventos o notificaciones por *syslog*. El sistema de trazas consta de un aplicativo servidor que recibe los

[64] **Clave foránea.** *Foreign Key – FK.* Identifica una relación entre tablas de la base de datos.

mensajes de DataPower. El servidor se encuentra en la misma red local que DataPower y es capaz de recibir los logs de DataPower mediante UDP[65].

El funcionamiento es el siguiente: dentro de DataPower se generan constantemente todo tipo de logs definidos cada uno por el campo *category*, que define el grupo o ámbito en el que se genera la alerta, por el campo *priority*, que define la prioridad, y por otro campo *service code*, que define el código específico del error. Por lo tanto, la consecución de estos tres parámetros determina un tipo de evento concreto. En este concepto es cuando entran en juego el concepto *Log Target*. Éste es un objeto de DataPower que gestiona el tratamiento de los logs de sistema mediante suscripción y publicación. En concreto, este tratamiento es realizado mediante un método de suscripción a los eventos de sistema que se van generando, para luego darles un destino a cada uno (fichero local, FTP, *syslog*, etc.). Dicha suscripción es muy flexible; por ejemplo, permite que existan varios objetos *Log Target* con distinto filtrado de "*category/priority/service code"* y con distinto destino, esto es, varios tratamientos de logs de manera simultánea.

Por defecto en DataPower siempre existe un *Log Target* suscrito a todos los eventos y con destino un fichero en un directorio local. Normalmente va a ser insuficiente, por lo que, en nuestro caso, se crea un nuevo *Log Target* para suscribirnos a los eventos del grupo *all* (cualquier tipo de evento generado por DataPower) y de mayor o igual nivel que *notice*. Se descarta así automáticamente los eventos de menor prioridad: *information* y *debug* por generar un volumen de trazas muy alto y no contener información relevante de almacenar. El destino de dichos eventos de alta prioridad va a ser el servidor externo, recogiendo los datos con mediante *syslog* y almacenándolos en un fichero de texto plano.

A continuación, se describe la prioridad (*priority)* de los mensajes que se van a publicar en el servidor remoto ordenados de menor a mayor. Se muestran también trazas de DataPower para los ejemplos desarrollados:

- *Notice:* notificación que normalmente no requiere una acción asociada.

```
        DP_SERGIO [0x00350014][mgmt][notice] source-https(HTTPs_WS_PROXY_IN): trans(111):
Operational state up
```

- *Warning*: advertencias que, sin ser errores graves, deben corregirse.

```
        DP_SERGIO [0x80400021][ws-proxy][warn] xmlmgr(default): trans(29183): Compilation
warning: Warning at local:///IDServiceBinding.WSDL:30: soapAction is required for any operation
using the HTTP transport binding of soap
```

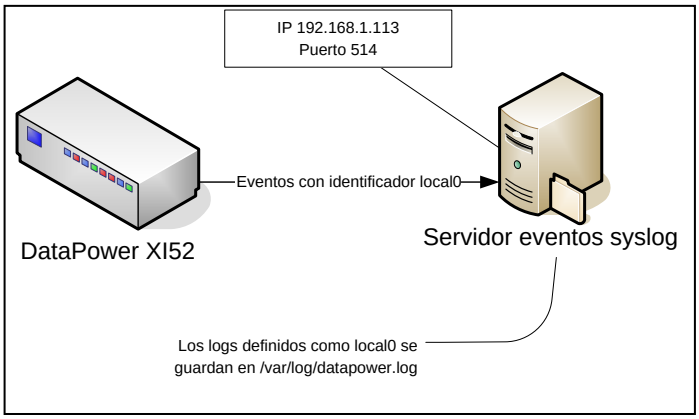
- *Error*: errores en aplicaciones que pueden afectar a su funcionamiento.

```
        DP_SERGIO [0x80e00625][network][error] trans(27583): Connect to URL
'ftp://ftp.orange.es/?Type=raw-netascii' timed out
```

- *Critical*: el error es crítico y requiere intervención.
- *Alert*: similar al anterior pero un nivel más elevado de prioridad.
- *Emergency*: nivel máximo de criticidad.

Los mensajes de tipo *critical*, *alert* y *emergency* requieren una intervención inmediata, siendo el mensaje de tipo *emergency* indicativo de que el sistema está completamente inoperable.

[65] **UDP:** *User Datagram Protocol*. Protocolo de la capa de transporte del modelo TCP/IP no orientado a conexión, por lo tanto no proporciona control de errores pero genera una carga de paquetes menor.



[Figura 3.5]: Diagrama resumen de la comunicación de eventos a servidor

Para diferenciar los eventos de sistema internos del propio servidor Ubuntu y los que se envían desde DataPower, se utiliza el campo *facility*. Es similar a un contenedor de eventos que, para nuestro desarrollo, hay que definir en DataPower y en la máquina Ubuntu. El valor concreto del campo para nuestros ejemplos es *local0.* Por lo tanto, se debe configurar en el servidor externo un tratamiento de eventos consistente en: almacenar alertas con un determinado *facility* en un fichero de texto y realizar rotado y purgado de manera automática por el demonio *rsyslog* de Ubuntu.

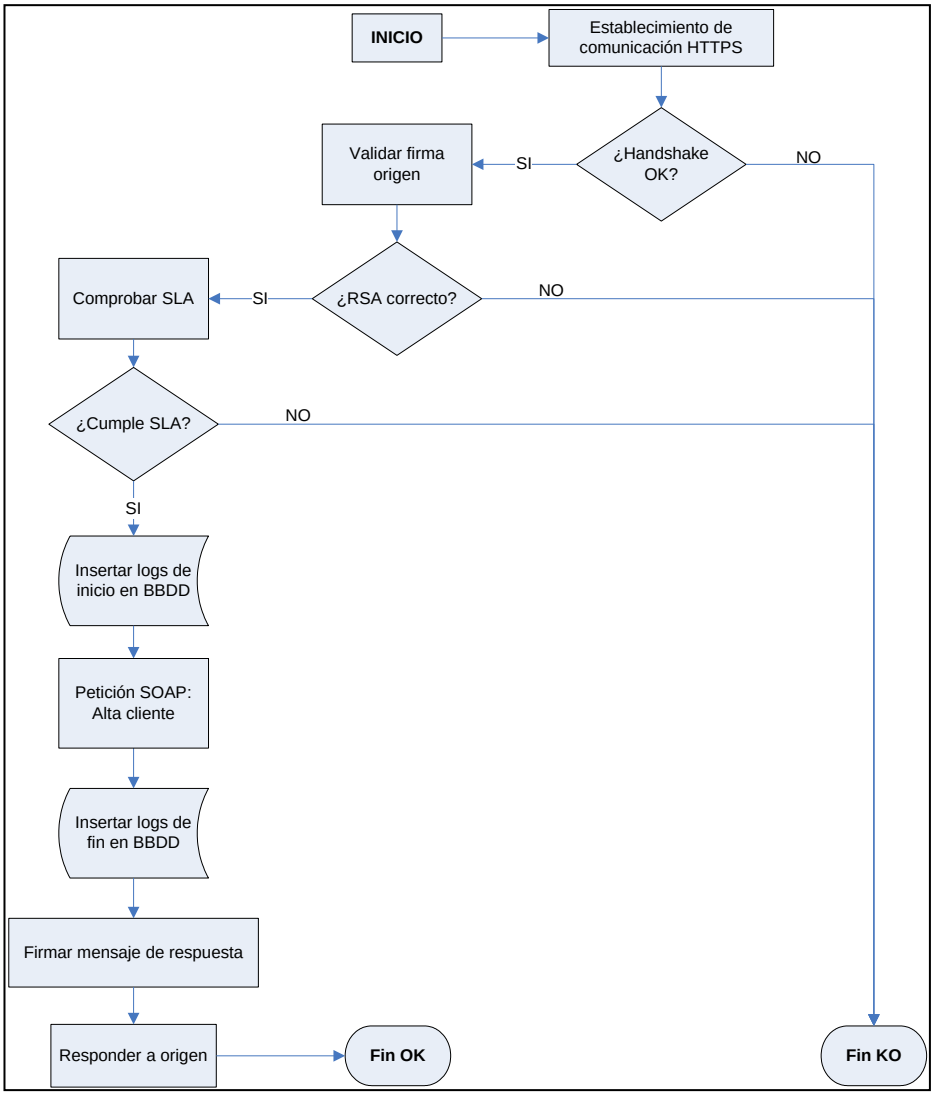
3.4.3.2 Diseño de DataPower

El diseño de los servicios web en DataPower para una comunicación extremo a extremo debe diferenciarse dependiendo del sentido de la comunicación, ya que la forma de actuar de cada uno es distinta.

a) Comunicación extremo a extremo: sentido entrada

En la [figura 3.6] se expone el diagrama de estados que detalla la implementación funcional del ejemplo de servicio web en sentido entrada. El evento de inicio lo desencadena una llamada HTTPs desde la Empresa B a la Empresa A, que pasa por DataPower a través de la interfaz *eth1*.

El flujo consta de las acciones que se describen a continuación. En primer lugar, se verifica si el origen es un cliente permitido en nuestro servicio: se realiza el *handshake* para la comunicación HTTPs. Para gestionarlo se utiliza el objeto de DataPower *Crypto Validation Credentials*, el cual representa un grupo de certificados que identifica a una determinada entidad. Este objeto debe coincidir, en nuestro caso, con el certificado de la Empresa B. En caso negativo, se descarta la petición sin incluir ningún dato en el log de sistema, puesto que algún ataque externo de un sistema no verificado podría saturar la base de datos al realizar peticiones concurrentes para su inserción. Posteriormente se valida con el mismo objeto *Validation* si el RSA generado por la Empresa B dentro de la firma del mensaje es correcto. En caso contrario, se desecha la petición de la misma manera que antes.



[Figura 3.6]: Diagrama de estados de un servicio web de entrada

Posteriormente se realiza una verificación del SLA ligado a dicho servicio web. Esta verificación lo que va a realizar es comprobar que el número de peticiones recibidas no supere un cierto umbral en un determinado número de segundos. En el caso en que se supere, la petición se rechaza y no se procesa.

Tras la verificación del SLA, en caso de que no se sobrepasen los umbrales, se inserta la petición de inicio en base de datos. En este caso se inserta sólo la petición de inicio, por lo que sólo los campos mostrados en la [tabla 3.2] son los que van a rellenarse para la tabla TRAZAS.

TRANSACCION	SERVICIO	DISPOSITIVO	FECHA_INICIO	FECHA_FIN	CODIGO_RESUL	DESC_RESUL	IP_ORIGEN
X	X	X	X				X

[Tabla 3.2]: Parámetros insertados en la petición de un servicio web

En lo que respecta a la tabla DATOS, ésta se rellenará con los datos de la llamada a los servicios web. Los datos a insertar variarán dependiendo de la ejecución, por lo que el

desarrollo del tratamiento de los mismos debe realizarse siempre de manera personalizada. Esto es aplicable a todos los servicios web implementados que contengan tratamiento de trazas en DataPower.

Como se comentó antes, no es objeto del proyecto el desarrollar un servicio web interno que ofrezca una funcionalidad real, sino más bien el proporcionar a dicho servicio de la empresa (en teoría ya existente) de una comunicación con un sistema de otra red mediante DataPower. Por lo tanto, se simplifica el diseño de la llamada a la aplicación interna.

En el caso de una comunicación de entrada a nuestra empresa (*IN*) la llamada al sistema interno va a ser la del alta de un nuevo cliente. Consta de un solo dato de entrada, un identificador; debe constar de 2 letras [A-Z]. La respuesta constará de una copia del mismo identificador seguido del carácter `-` y de una lista de 5 dígitos a modo de contador. Por lo tanto, es un caso sencillo en el que la respuesta sólo posee un único dato. Además, no se implementa en DataPower una gestión de errores.

Una vez realizada la llamada al servicio web, se debe volver a guardar la transacción en el sistema de logs. En este caso se debe guardar sobre el mismo registro anterior (ayudándonos del número de transacción de DataPower) los nuevos datos a insertar y que anteriormente quedaron vacíos.

TRANSACCION	SERVICIO	DISPOSITIVO	FECHA_INICIO	FECHA_FIN	CODIGO_RESUL	DESC_RESUL	IP_ORIGEN
				x	X	x	

[Tabla 3.3]: Parámetros insertados en la respuesta de un servicio web

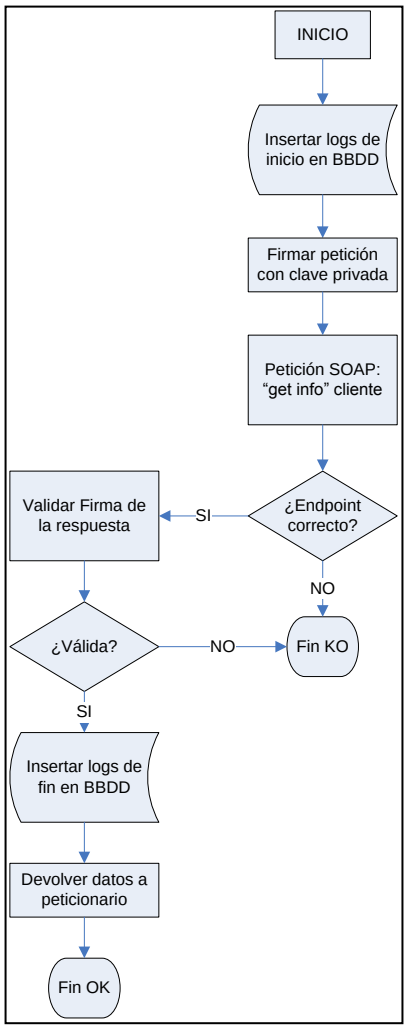
Finalmente, antes de responder al destino, se debe realizar un firmado de los resultados utilizando nuestra identidad. Se utiliza para ello un objeto de tipo *crypto identification credential*, el cual consta de una clave y de un certificado emparejado. En este caso el certificado pertenecerá al de la Empresa A.

Respecto al tratamiento de errores, como se puede ver, no existe salvo los derivados de una identificación incorrecta o de un incumplimiento en el SLA.

b) Comunicación extremo a extremo: sentido salida

Para la situación de una comunicación punto a punto en sentido salida, el procedimiento es quizá más sencillo, ya que ni se requiere verificación de la identidad del llamante ni verificación de SLA.

El diagrama de estados de un servicio web en sentido salida se puede observar en la [figura 3.7]. En este proceso, el inicio de la ejecución se desencadena con la llamada a un servicio web mediante HTTP a la interfaz *eth0*, ya que el origen está en la misma red de confianza, desde la Empresa A hacia la B.



[Figura 3.7]: Diagrama de estados de un servicio web de salida

En el diagrama anterior se puede observar que, para el caso del ejemplo que se va a implementar de comunicación entre dos empresas, la llamada SOAP que a realizar es la de obtener datos de un cliente en concreto del sistema externo. La misma se implementa utilizando un *end-point* estático mediante HTTPs, es decir, es un destino fijo en la red externa. Además, en la llamada, nos identificaremos con un objeto de tipo *crypto identification credential* para la Empresa A y validaremos mediante un *crypto validation credential* de la Empresa B. Si no se realizara esta verificación, un atacante podría recopilar los datos que nosotros enviemos al *end-point* haciéndose pasar por el destino correcto. Esto es conocido como un ataque de tipo *Man in the Middle*[66]. El significado de los objetos de DataPower descritos aquí, se explicarán con más detalle en el apartado de la implementación del servicio.

La inserción de logs se trata de manera similar al del servicio web de entrada descrito en el punto anterior, utilizando las mismas tablas del modelo de datos pero insertando otro nombre de servicio web en el campo TRAZAS.SERVICIO. Por lo que se pueden reutilizar parte de los recursos o aplicaciones creados para otro servicio, por lo que no aplica únicamente a uno en concreto.

[66] **Man in the Middle:** se basa en que el atacante intercede en la comunicación entre los pares sin que los mismos acusen su presencia, siendo capaz además de interceptar la información confidencial que comparten.

Finalmente, similar al servicio web anterior, éste no cuenta con tratamiento definido en caso de error. Dado que no se implementa, si existiera un error en cualquiera de los puntos del flujo, DataPower pararía la ejecución del mismo. El error no se guardaría en base de datos y no quedaría registrado.

En caso de necesitar analizar las trazas en detalle de una determinada ejecución, DataPower proporciona una herramienta dedicada para tal fin. Mediante la misma se pueden obtener las trazas completas de ejecución, además de sus variables de proceso, y detectar el punto concreto del fallo prácticamente en tiempo real. Este modo especial se conoce como *Probe*[67]. El detalle de las trazas en base de datos no implica necesariamente que se determine con exactitud el tipo de fallo que esté ocurriendo. Sí que nos van a servir para detectarlo y poder realizar informes de volumen a posteriori.

3.4.4 Implementación del servicio en DataPower

Debido a tener dos diseños distintos para una comunicación extremo a extremo dependiendo del sentido de la misma, se deben realizar dos implementaciones distintas en DataPower.

A continuación se describen las etapas de los WS de entrada y salida que para ambos escenarios son iguales en continente pero no en contenido:

- Definición del WSDL de la comunicación
- Creación de un objeto *Web Service Proxy*[68] (WSP) en DataPower
- Definición del destino del WS de DataPower
- Definición del origen del WS de DataPower
- Reglas y tratamiento de las transacciones que se tramitan por el WS
- Gestión de la firma de las transacciones
- Definición de las mediciones del SLA del WS
- Implementación de las trazas del WS en Base de Datos

3.4.4.1 Comunicación extremo a extremo: sentido entrada

a) Definición del WSDL

El primer paso a realizar para crear el *Web Service Proxy* es crear u obtener el WSDL para definir el comportamiento del servicio. DataPower en estos casos actúa como un *proxy* que añade lógica adicional. El fichero define los parámetros de entrada y salida, su formato, las operaciones implementadas y el destino. Al actuar como *proxy*, el WSDL a utilizar es el del servicio de destino de la comunicación. Lo cual significa que DataPower normalmente se va a comportar, en términos de intercambio de parámetros entre origen y destino, como el servicio web remoto, ya que se comporta como una capa adicional que proporciona seguridad.

[67] **Probe:** herramienta de DataPower que permite capturar la información que se envía y recibe para un determinado servicio, proporcionando además visibilidad de las variables de entorno y ejecución que haya en cada uno de los pasos de tratamiento de las reglas. Es capaz de mostrar toda la información de la transacción, en formato texto, a la entrada y salida de cualquier acción para facilitar su interpretación. Se realiza a nivel de objeto de servicio, es decir, a nivel de *Web Service Proxy* o *Multi-Protocol Gateway*. Este modo *debug* implica un coste operacional de DataPower, por lo que se recomienda desactivarlo si no es estrictamente necesario.

[68] En el capítulo 10 de la referencia [46] se define el objeto *Web Service Proxy* de DataPower.

Para todos los servicios web implementados, la definición del WSDL se ha recuperado de un directorio público[69]. El motivo ha sido meramente práctico: se adaptaba a las necesidades de diseño y además ese mismo WS se expone directamente a cualquiera que necesite experimentar[70]. Todo ello implica que sean procesos que estén lo suficientemente probados y no existan fallos en la definición de tipos de datos en el WSDL. En caso de que se opte por el método de crearlo desde cero, se disponen de diversas plantillas proporcionadas por la W3C[71]. Para el caso concreto de este apartado se obtiene el fichero *IDServiceBinding.WSDL* de un directorio público.

Funcionalmente, el WS pone a disposición del usuario un proceso que simula el alta o creación de un cliente en un sistema de nuestra empresa. El ejemplo seleccionado va a simular la generación de un identificador de usuario dada una entrada de datos. Como parámetro de entrada al servicio se define un tipo de dato *String*. La salida va a ser un tipo de dato a medida y que se compone de dos caracteres seguidos de un guion y cinco dígitos. En la siguiente tabla se muestra el resumen de los campos de entrada y de salida del servicio web implementado en DataPower:

ENTRADA	
Nombre	**Tipo de dato**
generate	String
SALIDA	
Nombre	**Tipo de dato**
IdentifierType	[A-Z]{2}-\d{5}

[Tabla 3.4]: estructura del WSDL de WS-Proxy en sentido entrada

En muchas ocasiones los WSDL utilizados cargan dependencias de otros ficheros externos. Para nuestro WSDL en particular se ha recuperado un archivo de definición que importa varios ficheros en formato XSD (*XML Schema Definition*) con la estructura de los datos. Se cargan normalmente con los siguientes procedimientos:

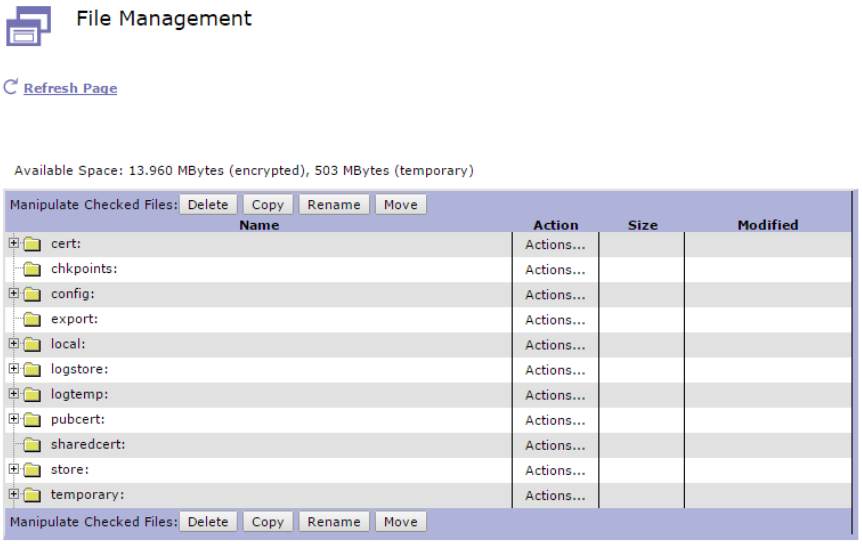
```
<xsd:include schemaLocation="location"/>
<xsd:import schemaLocation="location" namespace="ns"/>
```

Dichos ficheros van a almacenarse en el directorio *local:///* de DataPower. Este directorio siempre existe por defecto para cada dominio y no es compartido entre los mismos. Es usado normalmente para almacenar datos de usuario de uso general, exceptuando certificados y claves.

Como ya se ha comentado, las claves y certificados se almacenan automáticamente en el directorio *cert:///*. Los certificados públicos almacenados aquí pueden extraerse y consultar sin problemas. Sin embargo, las claves privadas no se pueden extraer de ningún modo. Como ya se ha comentado, para volver a utilizarlas a fin de crear un nuevo certificado, se puede utilizar las *Crypto Tools* de DataPower.

[69] Se obtienen los WSDL desde la URL [47]
[70] El WSDL original se puede recuperar de [48]. Este WSDL contiene el método *IDService* para la creación de un cliente. Dentro del repositorio se pueden realizar pruebas de comportamiento a través del mismo navegador web y comprobar los datos devueltos. Enlace del simulador web con parámetro la URL del WSDL a probar: [49]
[71] Plantilla para WSDL v2.0: [50]

En la [figura 3.8] se puede ver la estructura general de los directorios de DataPower (*"Administration>Main>File Management")* en una captura obtenida de la propia consola de gestión del aplicativo.



[Figura 3.8]: Estructura de directorios de DataPower XI52

Dado que todos los ficheros referenciados por el WSDL van a guardarse en DataPower, la referencia en el código debe ser modificada para apuntar a *local:///.* Este cambio se realiza modificando las líneas con la ruta absoluta de DataPower o simplemente con el nombre del fichero si está guardado en la raíz del directorio *local*:

```
<xsd:include schemaLocation="local:///fichero.xsd"/>
<xsd:import schemaLocation="local:///fichero.xsd" namespace="ns"/>
<xsd:import schemaLocation="fichero.xsd" namespace="ns"/>
```

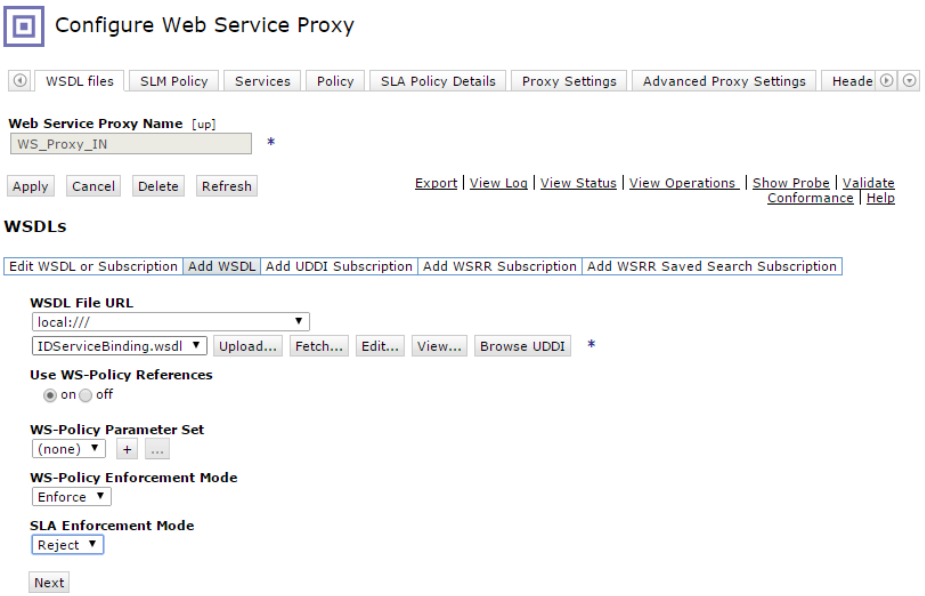
b) Creación del objeto Web Service Proxy

Una vez tengamos, de cualquiera de los modos anteriores, la definición del servicio WSDL accesible en DataPower, hay que cargarlo en un nuevo objeto de tipo *Web Service Proxy*. Para crear uno nuevo se puede seleccionar en el menú *"Objects>Service Configuration>Web Service Proxy"* o desde la ventana principal seleccionando a través del menú *Control Panel* la opción *Web Service Proxy* y posteriormente *Add*. Una vez creado el objeto, se le pueden añadir al mismo tantos WSDL como se quiera. Lo más sencillo y más lógico para los desarrollos es utilizar componentes individuales para cada implementación. La mayor ventaja es la independencia entre los desarrollos de distinto *Web Service Proxy.* Teniendo en mente lo anterior, se crea el servicio web seleccionando el WSDL de *local:///IDServiceBinding.WSDL*.

Las opciones relativas a *WS-Policy* se dejan por defecto dado que no se van a utilizar. Consiste en definir una política de seguridad en *WS-Policy Parameter Set* mediante el cual se verifican las peticiones de cliente o las respuestas de un servidor. Dependiendo de *WS-Policy Enforcement Mode*, con valores como *filter* y *enforce*, se actúa en caso de que no se cumpla. Con *filter* se está definiendo que cualquier mensaje que no satisfaga la política en cuanto a temas de seguridad, se deseche. En cambio, con *enforce* se fuerza la respuesta, modificando el mensaje e incluso añadiendo campos que falten si se han definido antes para

que cumpla las condiciones. Se dejan por defecto ya que no va a ser necesario utilizarlo en nuestro desarrollo.

En la [figura 3.9] se muestra la captura de consola de la ventana de creación.



[Figura 3.9]: Creación de un Web Service Proxy en DataPower XI52

La opción *SLA Enforcement Mode* se modifica a *Reject*. Con esto, se incluye un módulo adicional, configurable en la sección de las políticas de tratamiento de las peticiones (*Policy*) para verificar el acuerdo de nivel de servicio.

c) Definición del destino (Remote Endpoint) - HTTP

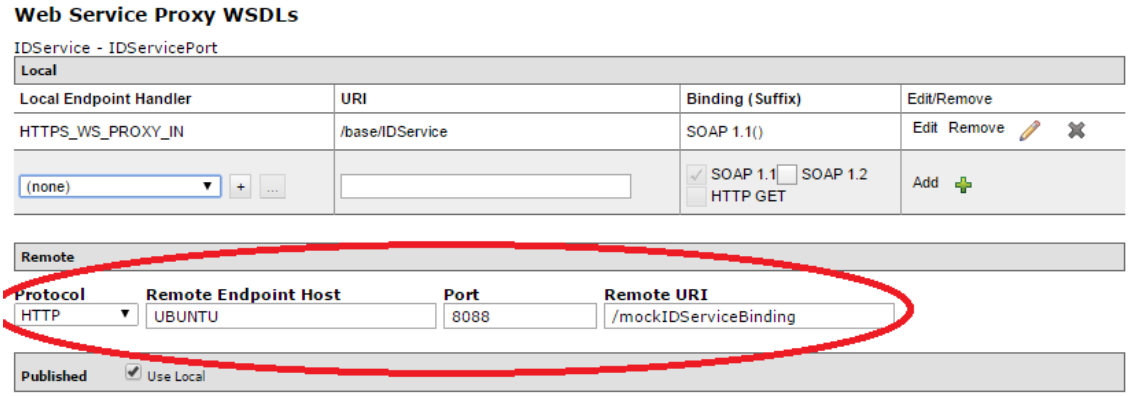
El siguiente paso es definir el origen y el destino del servicio, o lo que es lo mismo, el *Local Endpoint Handler* y el *Remote Endpoint Host*. Para explicar este concepto nos podemos remitir a la [figura 3.2], en la que se observa un flujo de comunicación para un servicio web en sentido entrada marcado en color azul. A modo de resumen:

- *Local Endpoint* se debe definir una dirección HTTPs con la IP de la interfaz *eth1*, el punto de entrada al servicio implementado en DataPower.
- *Remote Endpoint* se va a utilizar una aplicación de nuestra red interna sita en la máquina Ubuntu.

En lo que respecta a la configuración del *Remote Endpoint* o destino, se va a levantar en el servidor Ubuntu un autómata que devuelva una respuesta fija ante un mensaje SOAP determinado: se utiliza un *Mock Service* en SoapUI dado que es una opción realmente sencilla para las pruebas. El proceso funciona de manera análoga a un servicio web pero sin ninguna lógica detrás.

Para referenciar al *Mock Service* se define su IP mediante un *alias* ("UBUNTU") y su respectivo puerto, 8088, junto con su URI o ruta de entrada, */mockIDServiceBinding*. En dicha figura también se puede ver la configuración relativa al punto de entrada a la red o *Local Endpoint* de la que se hablará más adelante.

El *Endpoint* se muestra en la [figura 3.10] en el apartado llamado *Remote*.



[Figura 3.10]: Definición de local endpoint y remote endpoint en un Web Service Proxy sentido entrada

Respecto al *alias* señalar que es muy recomendable definirlos de modo habitual para no apuntar directamente a las direcciones IP de las máquinas. Una modificación de la misma implicaría la revisión de todo el código, sin embargo, con un *alias* sólo se cambiaría este "*Known Host*" en la configuración de DataPower, por lo que las aplicaciones podrían conectarse sin problemas a la nueva IP.

El *Remote EndPoint* del servicio se define también en el WSDL bajo la cláusula:

```
<service name="IDService">
          <port name="IDServicePort" binding="tns:IDServiceBinding">
                    <soap:address location="http://host:port/URI">
          </port>
</service>
```

Cuando se crea un *Web Service Proxy*, automáticamente el *backend* definido en DataPower va a ser el descrito dentro del propio WSDL. Por lo tanto, se debe modificar en la consola manualmente por la IP destino. Como se ha comentado antes, para este *Web Service Proxy* es "UBUNTU", sita en la red local y que tiene levantado un proceso *Mock* para proporcionar una respuesta.

Para que esta configuración sea posible es necesario marcar el campo "*Proxy Settings>Type*" al uno de los valores: "*Static from WSDL*" o "*Static Backend*". Se ha elegido la primera opción para configurarlo a nivel de WSDL. Lo que va a hacer DataPower va a ser sustituir, en tiempo de ejecución, el destino del WSDL por el indicado como *Endpoint*. Si se hubiera elegido la otra alternativa, se debería definir un *backend URL* al cual se enviarían todas las peticiones. La diferencia entre una y otra radica en que mientras que la primera se realiza a nivel de WSDL, la segunda se realiza a nivel de *Web Service Proxy*. Por lo tanto, en este último caso, si bajo el mismo *Web Service Proxy* se hubieran definido varios WSDL, todos se utilizarían el mismo destino independientemente de lo que tuvieran configurado en el WSDL. Para nuestro ejemplo, esto sólo tiene diferencias en el modo en el que se gestiona internamente, ya que únicamente va a existir un WSDL por cada *Web Service Proxy* creado.

d) Definición del origen (Local Endpoint) - HTTPs

Para el *Local Endpoint* es necesario realizar una tarea un poco más compleja puesto que es un enlace HTTPs y hay que definir los certificados de la comunicación. Como se puede observar en la figura anterior, en la parte titulada como *Local*, se ha definido un destino

llamado "*HTTPs_WS_PROXY_IN*". Éste es un objeto de tipo *HTTPs Front Side Handler* o lo que es lo mismo, un objeto para definir un punto de entrada utilizando un protocolo HTTPs. Las características principales del objeto son las mostradas en la [figura 3.11].

Name	Status	Op-State	Logs	Administrative state	Comments	Local IP address	Port	SSL proxy profile	Quiesce State
HTTPS_WS_PROXY_IN	saved	up	🔎	enabled	FSH para WS PROXY IN	DATAPOWER_EXTERNA	443	SSL_Profile_WS_PROXY_IN	

[Figura 3.11]: Resumen de características para un objeto HTTPs Front Side Handler

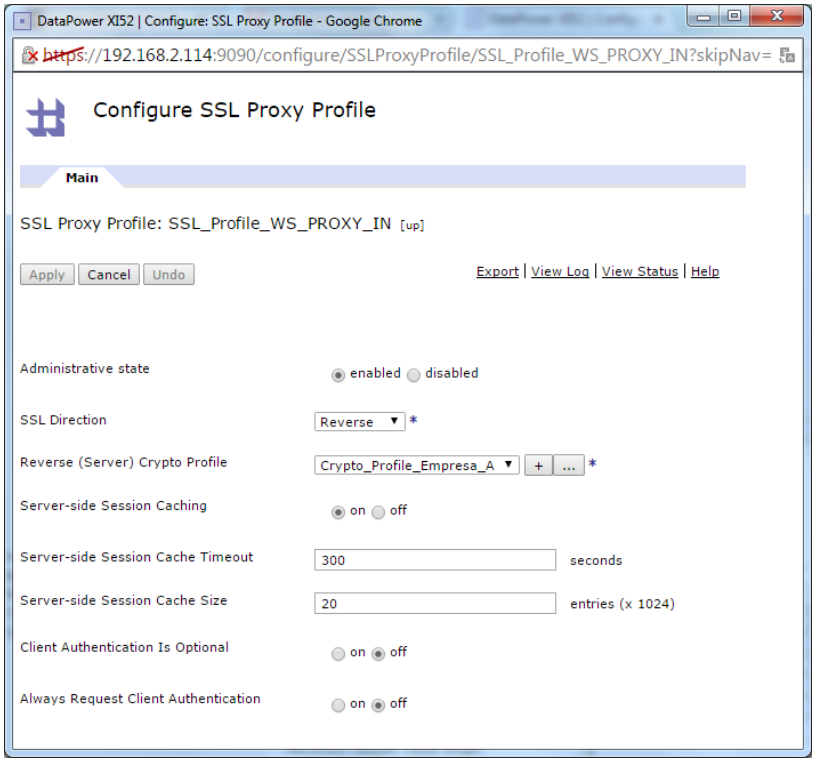
Para definir este tipo de objetos es necesario principalmente definir una IP, un puerto y un objeto de tipo *SSL Proxy Profile.* La IP que hay que definir es la relativa al *eth1* ya que es la interfaz dedicada a las comunicaciones desde el exterior. Como en el caso del servidor "UBUNTU", también se ha definido un *Known Host* de nombre *"DATAPOWER_EXTERNA"*. El puerto es el 443, por defecto para las comunicaciones seguras. Realmente se podría utilizar cualquiera que estuviera disponible mientras esté abierto en nuestro *Gateway*.

El resto de opciones se dejan por defecto. Las mismas consisten en la seleccionar los métodos disponibles para dicho objeto (PUT, GET, POST, HEAD, DELETE, etc.), la versión de HTTP (1.0-1.1), las conexiones persistentes (que por defecto están habilitadas), la codificación de caracteres (por defecto definida por el protocolo), opciones relacionadas con el tamaño y número de cabeceras de la trama y un control de acceso por IP: se puede establecer un listado de IP de origen a las que sólo se les permite ejecutar el servicio, aunque no se va a utilizar en los desarrollos, es importante para poder restringir más aún las comunicaciones si se sabe con seguridad las IP con las que nos llaman.

El siguiente paso es definir el objeto *SSL Proxy Profile* que se utilizará para establecer la comunicación HTTPs. A continuación se desgrana la información de las características relevantes para uno de ellos:

- *SSL Direction*: relativo al sentido de comunicación del protocolo SSL. Valores:
  - *Forward*: DataPower se comporta como un cliente, realizando la petición de conexión a un servidor externo.
  - *Reverse*: DataPower actúa como un servidor SSL, recibiendo peticiones de clientes. Esta es la opción que se va a utilizar ya que el *Web Service Proxy* es en sentido entrada, hacia nuestra Empresa A.
  - *Two-Way*: actúa como cliente y servidor, realizando de proxy de comunicaciones aceptando tráfico SSL sobre TCP y creando nuevas conexiones SSL sobre TCP hacia el destino.
- *Client Authentication Is Optional*: esta opción está disponible solo cuando DataPower actúa como servidor SSL. Con ella se fuerza a que, cualquier cliente que acceda a DataPower utilizando el objeto que define este protocolo SSL, deba autenticarse con un certificado válido. Si se dejara como opcional, el servicio finalmente estaría disponible para cualquier usuario que lo ejecutara.
- *Always Request Client Authentication*: Cuando actúa como servidor, siempre pide las credenciales del cliente incluso si no se ha definido un objeto *Validation Credential* con el que cotejarlas. Para nuestro propósito hay que definir ese objeto así que esta propiedad puede tomar cualquier valor ya que define el comportamiento en caso de que no exista

En la [figura 3.12] se muestra su ventana de configuración de la consola de DataPower para un objeto *SSL Proxy Profile*.



[Figura 3.12]: Definición de un objeto SSL Proxy Profile

Para completar el *SSL Proxy Profile* hay que definir un objeto de tipo *Crypto Profile*, que especifica los detalles de la conexión SSL. Consiste básicamente en especificar en qué va a consistir el *handshake* o negociación inicial para una comunicación basada en cifrado asimétrico (clave y certificado).

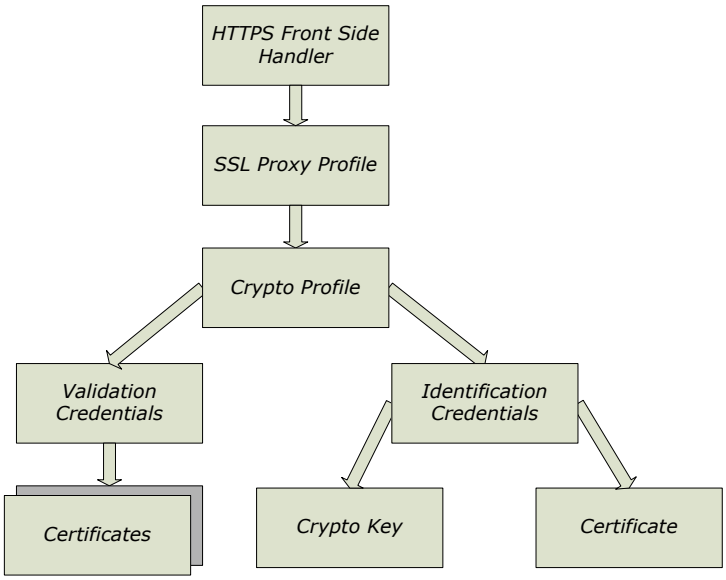
Durante la negociación inicial hay que definir el protocolo de seguridad mediante un intercambio de especificaciones. En ese momento se indica qué protocolo se va a utilizar durante la totalidad de la comunicación, desde el más seguro TLS 1.2 hasta el más inseguro SSL Versión 2. El servidor siempre utiliza el que se ha definido como más seguro siempre y cuando el cliente también lo soporte. En caso de que el cliente requiera una versión inferior, ésta se tiene que negociar. Por defecto en DataPower se ofrecen los protocolos TLS 1.2, TLS 1.1, TLS 1.0 y SSL v3. El problema radica en que, si un cliente quiere hacer un ataque de tipo *downgrade* para aprovechar las vulnerabilidades del protocolo SSL v3, con la configuración por defecto lo va a poder realizar. Por lo que es necesario indicarle a DataPower los protocolos que va a permitir en ese intercambio de requisitos, deshabilitando el SSL y dejando sólo TLS en cualquiera de sus versiones.

Finalmente, para terminar de configurar nuestro *Crypto Profile* hay que definir como *Validation Credentials* a la Empresa B y como *Identification Credentials* a nosotros, la Empresa A.

Estos dos nuevos objetos, *Validation Credentials* e *Identification Credentials*, constan a su vez de nuevos elementos. En el caso del primero, hay que indicar los certificados de los clientes que hay que validar. Se puede indicar una lista de certificados, una cadena de confianza, que verifique si son todos válidos, que estén o no caducados, si se utiliza la CRL o Lista de Revocación de Certificados para no aceptarlos (similar a una lista negra). Para nuestro servicio sólo se va a incluir el certificado de la Empresa B creado con las *Crypto Tools*.

Respecto al objeto *Identification Credentials*, hay que configurar que nosotros como entidad seamos la Empresa A. En este caso es necesario indicar el certificado público y la clave privada. Ambos, como no, vuelven a definirse como dos objetos de DataPower, *Certificate* y *Crypto Key* respectivamente.

Una vez se hayan definido todos los objetos de la [figura 3.13] se puede dar por concluida la configuración del *Front Side Handler* para comunicaciones HTTPs.



[Figura 3.13]: Dependencia de objetos HTTPs

La monitorización proactiva de la caducidad de los certificados se configura bajo el dominio *default* en el objeto "*Objects>Crypto Configuration>Crypto Certificate Monitor"*. Por defecto está configurado para no deshabilitar los objetos que contengan certificados caducados. Además, DataPower va a publicar una alerta de tipo *warning* indicado que el certificado va a expirar dentro de 30 días. Ese log es recogido por el *Log Target* creado a tal efecto, el cual lo envía al servidor de apoyo por UDP como un evento *syslog*.

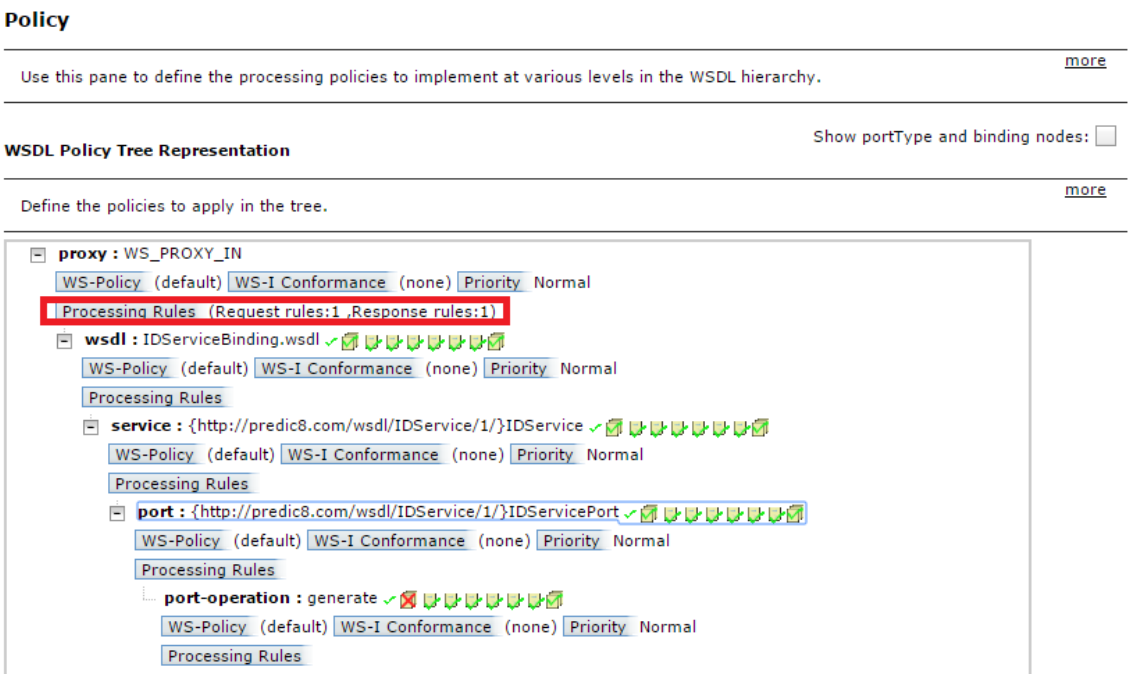
e) Reglas y tratamiento de transacciones

Una vez configurados el origen (*Front Side Handler*) y el destino (*Backend*) y la definición de la conexión HTTP/HTTPs para cada uno, se debe configurar la política de tratamiento de las transacciones. Se puede configurar a nivel de *Web Service Proxy*, como modo genérico para todo lo que tenga ese objeto definido, o más particularmente a nivel de WSDL, servicio, puerto u operación, dependiendo del nivel de profundidad de detalle en la operación a tratar. Nuestro desarrollo consta de una única definición de WSDL por objeto *Web Service Proxy* de DataPower, así que las políticas a generar se pueden realizar a nivel de proxy.

La [figura 3.14] muestra una captura de DataPower con las entidades del WSDL sobre las que se puede implementar una **política de tratamiento**[72]. Lo más común es generar dichas políticas a nivel de WSDL. En caso de tener un WSDL más complejo en el que se describan por ejemplo varias operaciones, se podría dar un tratamiento distinto a cada

[72] Las políticas de tratamiento o, como se llaman en DataPower, *Policy*, son acciones concatenadas para la gestión de las transacciones. Para ampliar información revisar el capítulo 7 de la referencia [46]

una creando las reglas a nivel de *port-operation*, ya que el WSDL tiene una estructura en árbol, desde lo general a lo particular. En la captura puede verse cada uno de esos niveles, comenzando a partir del *Web Service Proxy* y pasando por WSDL, *service, port* y *port-operation*. Cada uno de ellos además con su correspondiente *namespace*[73]. Para este caso el *namespace* pertenece al dominio *predict8.com.*



[Figura 3.14]: Policy en un Web Service Proxy

Las políticas de DataPower se componen de un conjunto de reglas de ejecución y definen el comportamiento que van a tener las transacciones. Las reglas pueden ser de varios tipos, dependiendo del sentido en el que se vayan a aplicar y de su uso.

Teniendo en cuenta que el rol de DataPower siempre es el de servidor:

- *Client to Server*: regla de la petición.
- *Server to client*: regla de la respuesta.
- *Both directions*: reutilizables para peticiones y respuestas.
- *Error*: captura el error y ejecuta la regla definida.

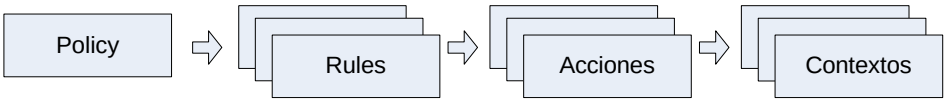
Una regla no es más es una sucesión de definiciones, transformaciones u operaciones que se aplican a cada mensaje que se procesa. Cada una de estas acciones de la regla tiene varios contextos de ejecución predefinidos. Un contexto es un conjunto de datos de ejecución, adjuntos MIME (*Multipurpose Internet Mail Extensions*) y variables de sistema y de usuario. Los contextos se definen en cada acción definiendo uno de entrada y otro para la salida. De este modo, cada acción realiza un tratamiento específico de los datos del contexto entrada para almacenarlos en el contexto de salida. Se pueden definir contextos personalizados para definir un tratamiento de las acciones en las reglas que no sea lineal. Los contextos predefinidos son:

- INPUT: definición de los datos de entrada a la regla.
- OUTPUT: define la salida final de la regla.

[73] **Namespace**: es un espacio conceptual que agrupa clases e identificadores para evitar conflictos con otros ítems con el mismo nombre. Se identifican con el atributo *xmlns:prefix="URI".*

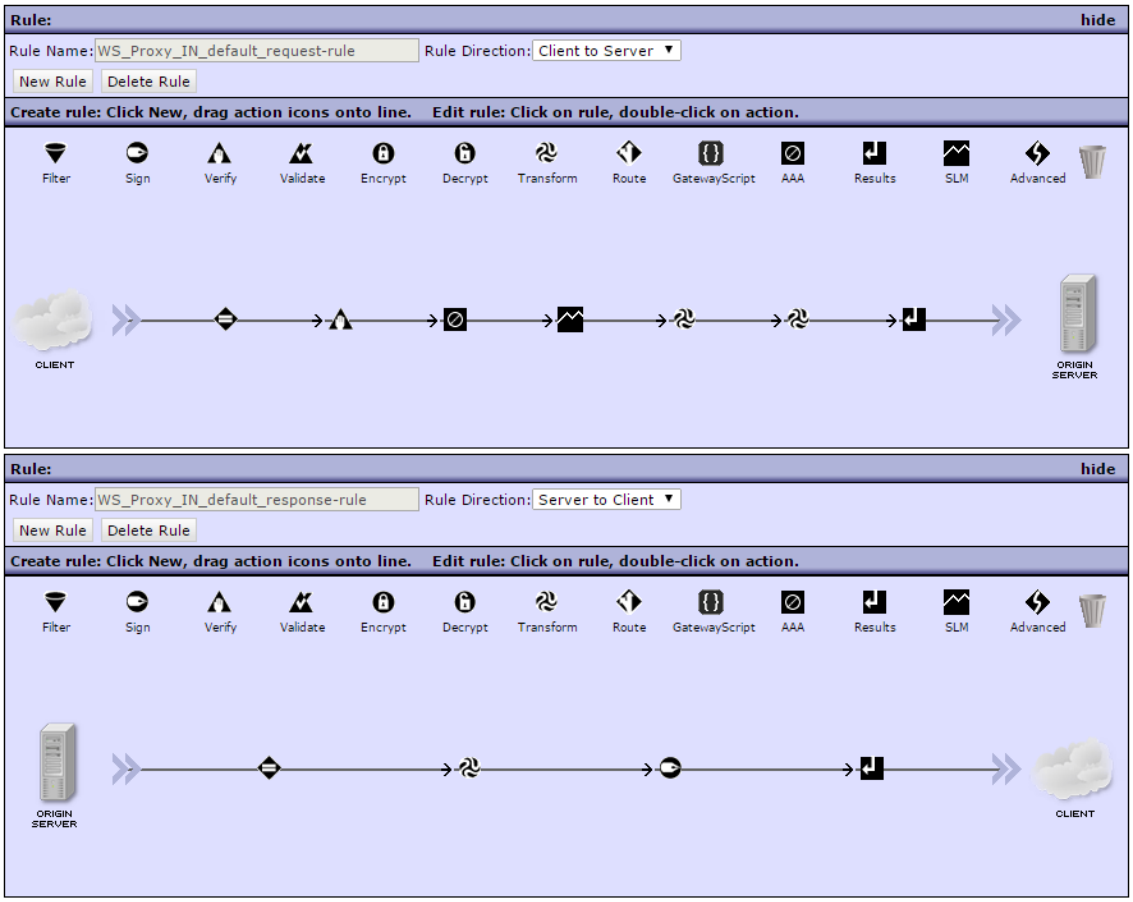
- PIPE: definición especial que provoca que la salida de una acción sea la entrada inmediata de la siguiente. Se utiliza por cuestiones de rendimiento y de tratamiento concatenado.
- NULL: no se utiliza contexto.

En la siguiente figura se muestra el resumen de los objetos que se han ido describiendo para la realización de las reglas de DataPower.



[Figura 3.15]: Resumen de definición de objeto *policy*

En lo relativo a las reglas generadas para este servicio, se muestra en la [figura3.16] la definición completa de las mismas.



[Figura 3.16]: Reglas para Comunicación extremo a extremo de entrada a DataPower

Nuestro servicio web en sentido entrada va a tener definidas dos reglas dependiendo del sentido de la transacción: petición o respuesta. No se define una regla de error, aunque siempre es muy recomendable implementarla para tener controlados los errores que ocurran

en la ejecución de la transacción por DataPower. En el apartado de pruebas y resultados se analizarán las consecuencias de no tenerla definida.

f) Gestión de la firma de las transacciones

Es necesario comprobar y realizar la firma para las peticiones y las respuestas de cara a aumentar la seguridad y la veracidad de la información. Al tratarse de un servicio web de entrada a la Empresa A, el mensaje a recibir siempre debe venir firmado por la Empresa B. Por lo tanto, en la regla de DataPower de la petición, en la [figura 3.16] de nombre "*WS_Proxy_IN_default_request-rule"*, se incluirá la validación de dicha firma. Se representa con el icono  [74]. De manera general, para definir las reglas en el entorno de desarrollo de DataPower, se arrastran las acciones predeterminadas a la línea de trabajo y se definen sus contextos para concretar el flujo de datos de cada una.

En esta validación se define un objeto de tipo *Validation Credential*[75] y el tipo de verificación de la firma digital para poder corroborar que el firmante de las peticiones es quien dice ser. Si estas comprobaciones no se incluyeran, un intruso podría alterar los datos enviados por el sistema externo.

Tras validar la firma, antes de enviar al destino los datos del XML recibido, es necesario eliminar las cabeceras SOAP que se han creado a tal efecto. Esto es posible gracias a una transformación XSLT predefinida del almacén de DataPower: *store:///strip-security-header.xsl.* Las transformaciones se indican en las reglas con el icono:  . Para que los datos resultantes sin cabecera de seguridad se envíen a destino, se utiliza el contexto especial PIPE para pasar directamente el resultado a la siguiente acción, que no es otra que la respuesta:  .

Para el caso de la regla de respuesta, dado que respondemos a la Empresa B con el resultado, necesitamos firmar el XML con las credenciales de la Empresa A. En la [figura 3.16] se puede observar, en la parte inferior con nombre *"WS_Proxy_IN_default_response-rule"*, la definición de la regla de respuesta.

La nueva acción a definir en este caso es la complementaria a la realizada en la regla de petición. Esta acción se define con el icono:  . Para la firma se deben definir dos objetos criptográficos, *key* y *certificate*, y el tipo de firma a utilizar (RSA). Como dichos objetos ya estaban creados de los pasos anteriores, se reutilizan las credenciales para firmar como la Empresa A.

g) Definición de las mediciones de SLA

El tercer paso que se muestra en el diagrama de flujo de la [figura 3.6] es verificar si la orden de ese cliente está cumpliendo el Acuerdo de Nivel de Servicio, ANS o SLA. Existe una acción predefinida en DataPower, dentro de la regla de la política, para poder medir unos SLA's en tiempo de ejecución: *SLM Action*[76] (*Service Level Monitoring*). En la [figura 3.16] corresponde con el icono señalado como  . Dentro de esta acción se especifican los parámetros a medir en un objeto llamado *SLM Policy*.

En nuestro desarrollo se van a implementar dos validaciones o *SLM Statement*:

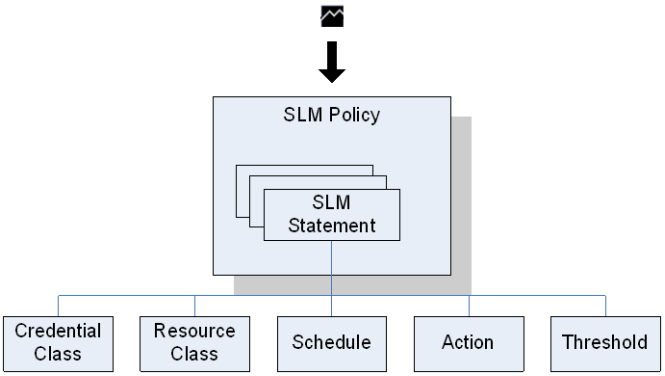
- Menos de 20 peticiones a la hora en un intervalo móvil.
- Un máximo de 200 peticiones al día en un intervalo fijo.

[74] La firma de las transacciones se desarrolla en el RedBook de IBM de la referencia [51]
[75] Ver [figura 3.13] con el árbol de objetos necesarios para crear un objeto de tipo *SSL Proxy*.
[76] Las reglas para gestionar los niveles de servicio se definen en la referencia [52].

La diferencia entre un intervalo móvil y uno fijo es que, mientras que para el fijo se define un inicio y un fin, para el móvil el intervalo es relativo a la posición actual. Esto es ventajoso en el caso de que para un tipo de intervalo fijo se superen desde el primer minuto a partir de su inicio el total de peticiones permitidas, lo que provocaría que el sistema no aceptara peticiones hasta que no comenzara un nuevo intervalo dentro de 59 minutos más.

Tenemos a nuestra disposición un amplio abanico de configuración para definir las políticas de tratamiento de nuestro SLM en DataPower. Para nuestro caso práctico, con la configuración de los dos niveles anteriores sería suficiente para evitar una posible saturación.

Las *SLM Statement* se componen de los objetos definidos en la [figura 3.17].



[Figura 3.17]: Partes de una acción de SLM

Es imprescindible realizar una configuración adecuada para cada objeto de con el fin de obtener la medición correcta de los niveles de servicio. Se describe a continuación la configuración realizada, aplicable además a otros casos:

- ***Credential Class***

Define qué clientes están sujetos a la ejecución de un determinado *SLM Statement*. Lo ideal es establecer que cada cliente esté definido por el certificado que utilice. Esto es, se configura para que cualquiera que utilice el certificado de la Empresa B sea representado como una única entidad, independientemente del número de servidores distintos que ejecute el WS desde esa empresa.

Para ello es necesario implementar una acción de tipo AAA (*Authentication And Authorization*) antes de la acción SLM para extraer los datos relativos a los certificados. En la [figura 3.16] está representado con el icono:  . Dentro de esta acción se debe implementar un mapeo de datos utilizando el método "*Credentials from WS-secureconversation token*", lo que extrae las credenciales.

Posteriormente es necesario configurar el objeto *Credential Class* como "*Mapped Credential*". Con esto se consigue mapear la información relativa a certificados de la petición de entrada extraída anteriormente en el objeto *Credential Class* de cara a identificar cada uno de ellos para aplicarles un SLM distinto.

- ***Resource Class***

Define la selección de los recursos que son recuperados por una determinada *Credential Class*. Para nuestra implementación se deja vacío, lo que equivale a aplicar a todos los recursos recopilados las acciones de SLM.

- ***Schedule***

Especifica un período de tiempo durante el que se va a aplicar el SLM. Para nuestro caso se define un período de 24x7.

- ***Action***

Define la acción a tomar cuando se encuentra una violación en el nivel límite configurado. Para nuestro propósito se configura para rechazarlas (*Reject*).

- ***Threshold***

Define el algoritmo a medir, indicando el método en el que se va a medir y los valores de referencia.

La configuración final es la que se muestra en la [tabla 3.5].

ID	Credential Class	Resource Class	Schedule	Action	Threshold Interval Length	Threshold Interval Type	Threshold Algorithm	Threshold Level
SLA limite por dia	Por certificado	(Vacío)	24x7	Reject	86400	Fixed	Greater Than	200
SLA limite en 60 minutos	Por certificado	(Vacío)	24x7	Reject	3600	Moving	Greater Than	20

[Tabla 3.5]: Resumen de configuración de las SLM Statement

h) Trazas de la transacción en Base de Datos

Un requisito a cumplir es el de la escritura de logs, necesario para auditorías a posteriori de las transacciones que cursa DataPower. Para ello, dado que el volumen de las peticiones puede ser muy alto y como se justificó en apartados anteriores, se optó por montar un sistema basado en un servidor con una base de datos para almacenar datos relativos a la transacción. Para realizarlo es necesario programar en XSL una transformación que se aplique al XML de entrada, aplicando, dentro de la misma, extensiones de DataPower para la gestión con bases de datos. Con ello se permite extraer los datos del mismo, guardarlos e insertar un registro del inicio. Esta acción se debe realizar tanto para el XML de la petición como para el XML de la respuesta SOAP.

El modo de realizar esta operativa en DataPower es realizando una acción de transformación en el tratamiento de las políticas de las transacciones. En la [figura 3.16] se representa con el icono:  Define una transformación de los parámetros de entrada basada en un fichero de transformación XSLT definido.

Para cumplir los requisitos expuestos anteriormente se define una transformación XSLT para el elemento del contexto INPUT de cada regla, petición y respuesta. El contexto INPUT va a representar la entrada de datos inicial a la regla, es decir, la que llega directamente a DataPower ya sea para la petición o la respuesta. Con ello se implementará el registro en la tabla TRAZAS y la inserción de los datos en la tabla DATOS.

La estructura del fichero utiliza la recursividad de la programación de esquemas o *templates* en XSL. El motor de procesado de XSLT no es un motor lineal, sino que utiliza la estructura de los elementos del fichero XML de entrada a transformar. Lo cual significa que

se recorre el árbol del XML de entrada según se seleccionan con un determinado *template*. Dentro del mismo están las acciones específicas a ejecutar. Por lo tanto, cada *template* definido en el fichero de transformación realiza una acción para los nodos del XML que coinciden con el *path* definido en el mismo, pudiendo además realizar llamadas dentro de los mismos para ejecutar otros esquemas (similar a llamar a una determinado método en otros lenguajes de programación como Java). Al final, el código no es más que es una sucesión de esquemas que definen diversas partes del XML a tratar.

El tratamiento de código XML en DataPower tiene muchas ventajas. Una de ellas es la velocidad, dado que es una máquina diseñada específicamente para este tipo de tareas. Otra no menos importante es el poder utilizar funciones y elementos específicos de DataPower, las llamadas *extension functions* y *extensions elements*. Las primeras se ejecutan dentro de las expresiones *XPath*[77] del código y las segundas son elementos que se ejecutan directamente en la hoja de estilo. Esto nos permite utilizar dentro del código XSLT funciones avanzadas tales como cifrar, firmar y verificar, validar certificados, autenticar mediante LDAP, ejecutar código SQL[78], validar un esquema XML, descodificar y codificar en BASE64[79], enviar una petición SOAP, etc. Respecto a las extensiones, las más comunes son las relativas a eliminar o añadir cabeceras HTTP, crear una variable de DataPower, definir el destino de la petición y realizar una llamada a una determinada URL. El listado completo de todas estas funciones y extensiones de DataPower se puede obtener de los *RedBooks*[80] que IBM proporciona libremente.

Para hacer uso de este tipo de facilidades, es necesario en primer lugar referenciarlas en el código XSLT para que DataPower permita su acceso. Como en nuestra solución es necesario realizar diversas operaciones con base de datos y crear variables modificables, se debe indicar en la cabecera del XSLT la utilización de las funciones y las extensiones de DataPower con el siguiente código:

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
          xmlns:dp="http://www.datapower.com/extensions"
          extension-element-prefixes="dp"
>
```

Para el caso práctico de este WS, se va a realizar la inserción de los campos del XML de la petición en dos tablas, TRAZAS y DATOS. Para esta última el número de filas a insertar depende del número de campos del XML de entrada. En el ejemplo, existe un solo parámetro a mapear en la tabla DATOS, esto es, el dato de entrada de tipo *String*.

Normalmente los XSLT comienzan procesando el inicio de la estructura XML de entrada, para nuestra solución va a ser así, se recorrerá desde la raíz hasta los campos determinados. Se le indica al motor de procesado que trate el nodo *root* de la estructura XML con el siguiente *tag*, que define el esquema o *template* principal:

```
<xsl:template match="/">
```

[77] **XPath:** es una definición para describir una localización con un documento XML. Permite direccionar elementos de un XML con un criterio específico, incluso referenciar el árbol XML en cualquier dirección. Es utilizada por estándares como XSLT, XPointer y XQuery.
[78] **SQL:** *Structured Query Language* – Lenguaje de consultas estructurado. Lenguaje utilizado para el manejo de bases de datos y que permite especificar diversos tipos de operaciones en ellas.
[79] **BASE64:** esquema de codificación para representar datos binarios mediante texto ASCII.
[80] Los ***RedBooks*** o documentos de IBM están disponibles en [53]

Posteriormente se definen las variables utilizando el contexto propio de DataPower en el que se definen las variables de sistema, *service*[81]. Por ejemplo, para el número de transacción, correspondiente a la *primary key* de cada tabla de base de datos, se utiliza el valor de la variable *transaction-id* obteniendo su valor mediante la función de DataPower "*dp:variable*":

```
<xsl:variable name="transactionId" select="dp:variable('var://service/transaction-id')"/>
```

Para la variable que contiene la IP del sistema llamante se utiliza una función de DataPower "*dp:http-request-header*". Ésta nos permite extraer los campos determinados de la cabecera HTTP[82] de la trama. Existen varios campos que nos proporcionan información de la IP del cliente, tales como *X-Forwarded-For*[83], *X-Client-IP*[84], etc. Sin embargo, para obtener la IP se carga el valor del campo *X-Client-IP.* Su elección viene condicionada a la disponibilidad de extracción de los campos que puedan contener la información requerida en la petición entrante. El listado de los mismos se muestra en la [tabla 3.6].

Content-Type	text/xml; charset=utf-8
SOAPAction	"/base/IDService"
User-Agent	Axis/1.4
Host	192.168.2.118:443
Content-Length	744
Via	1.1 BwAAAHUtFAE-
X-Client-IP	192.168.2.115
X-Global-Transaction-ID	3146840465

[Tabla 3.6]: Listado de cabeceras HTTP en la petición de entrada a DataPower

Además, el campo *X-Client-IP* se suele incluir una única IP origen, no un listado de IP's de los proxy por los que ha pasado, tal y como ocurre con el campo *X-Forwarded-For*. Al ser una comunicación entre empresas conocidas, no es necesario un detalle de las direcciones por las que ha pasado la petición, y dado que es un dato que se va a insertar en un campo de la base de datos, no conviene que sea de un tamaño indeterminado.

El modo de asignar el valor del campo *X-Client-IP* a una variable XSL es:

```
<xsl:variable name="clientIP" select="dp:http-request-header('X-Client-IP')"/>
```

Una peculiaridad de las variables de DataPower es que se pueden modificar, a diferencia de las variables de XSLT para las cuales no está permitido. Se va a utilizar una variable a modo de *token* para determinar si la inserción de los datos en la tabla TRAZAS es correcta. En caso afirmativo, para el tratamiento de la respuesta se podría realizar un *update*

[81] **Service** es un tipo de contexto de DataPower dentro del cual están definidas las variables de ejecución del sistema relacionadas con la transacción, el mensaje y el protocolo. Se puede encontrar desde variables relativas a la comunicación tales como *URL* destino, protocolo, etc., hasta variables de ejecución tales como código de error, transacción, etc. Las variables de este contexto son accesibles para toda la vida de la transacción (petición y respuesta) y puesto que son variables de DataPower, pueden ser modificadas, al contrario de lo que ocurre con las variables estándar de XSLT.

[82] En el capítulo 7.3.4 de la referencia [05] se puede observar la estructura de los campos de la cabecera HTTP.

[83] **X-Forwarded-For**: campo perteneciente a la cabecera HTTP utilizado para identificar la IP del servidor origen sin que sea transformada por un proxy. Su formato es *client1, proxy1, proxy2…*

[84] **X-Client-IP:** es un campo perteneciente a la cabecera HTTP que muestra la IP de origen de la petición. No es el dato más fiable a utilizar puesto que, si la petición pasa por un *proxy* intermedio, la dirección obtenida va a ser la de éste. Aun así, es el dato más cercano a la IP real que existe en DataPower.

del registro añadiendo los datos que faltan (fecha de fin y resultado). La definición de esta variable se realiza para un contexto propio, *"//context/bbdd/"*, y es visible para toda la vida de la transacción. Su definición es tal que así:

```
<dp:set-variable name="'var://context/bbdd/insertOKTrazas'" value="false()"/>
```

Para la configuración de la conexión se utiliza un parámetro de entrada con el objeto que define la IP, el puerto, el SID y usuario de la base de datos en DataPower (*SQL Data Source*). Este objeto se define en DataPower en *"Network > Other > SQL Data Source"*. Para el ejemplo que se está desarrollando se ha creado un objeto, nombrado "ORACLE_LOGS", que se conecta a una base de datos de tipo Oracle. Además, es necesario configurarlo como parámetro de entrada en la propia acción de transformación de DataPower. Al definir el objeto de entrada como "*dpconfig:DBConnection*", se indica a DataPower que no se refiere a un "literal" sino que se corresponde con el nombre de un objeto de ese tipo.

El código XSLT para definir el objeto de entrada y asignarlo a una variable es:

```
<xsl:param name="dpconfig:DBConnection"/>
```

En el caso de insertar las trazas en base de datos se utiliza la función *"sql-execute"* de DataPower. Como parámetros de entrada toma un objeto de tipo conexión a base de datos, en este caso la variable creada anteriormente, y un *String* con la sentencia SQL a ejecutar:

```
<dp:sql-execute source="$dpconfig:DBConnection" statement="$insertStringTrazas">
```

Una vez que se inserta en la tabla TRAZAS, se debe insertar en la tabla de DATOS. Para ello el modo de proceder siempre va a ser el mismo. Se estudia el XML de entrada para ir recorriendo los nodos hijos y en caso de estar en el final de un nodo, insertarlo en la tabla como se ha mencionado antes. El problema de esta configuración es que está realizada a medida y depende en cierto modo de la estructura de la petición y respuesta. Al utilizar un método recursivo para cargar los campos, si se modificara su número, no implicaría impacto en DataPower. Pero en caso de modificar la estructura general del código XML, se debería realizar un ajuste en el XSL para añadir dichos cambios. Normalmente suele solucionarse añadiendo una nueva plantilla que tenga en cuenta esa estructura.

Para empezar a trabajar y recorrer el nodo de manera recursiva, se utiliza la llamada a otros esquemas: *apply-template*. El motor busca, para el nodo en el que está del XML, el *template* a aplicar y lo ejecuta. Esto es, se va realizando un compendio de *templates* para ir recorriendo los distintos nodos. Dentro del *template* principal hay que comunicarle al motor que busque algún esquema coincidente con el nodo actual para aplicar las modificaciones:

```
<xsl:apply-templates/>
```

Posteriormente se definen los distintos *templates* para ir recorriendo la estructura. Si el nodo es un padre y no tiene los datos a insertar, se llama a otro *template*. Si es un nodo con los datos a insertar, en el *template* se realiza una llamada a la función de DataPower *dp:sql-execute*.

A continuación se muestra un ejemplo de estructura de inserción de datos realizada con plantillas o *templates* XSL.

```
<xsl:template match="*[local-name()='Envelope']">
          <xsl:apply-templates/>
</xsl:template>

…<xsl:template match="*[local-name()='generate']">
          <dp:sql-execute source="$dpconfig:DBConnection" statement="$insertStringDatosReq">
                    …
          </dp:sql-execute>
</xsl:template>
```

Por lo tanto, la estructura final del XSLT se basa en una reiteración a través de los elementos del XML de entrada. Para la primera iteración se inserta la traza de inicio de la petición en la tabla TRAZAS. Posteriormente, se recorre el árbol del XML hasta llegar a los nodos finales con los datos, y se insertan en la segunda tabla, llamada DATOS. Posteriormente para la respuesta, esta vez en sentido inverso de la comunicación, se debe insertar en la regla de la política otra transformación. Esta vez, en la primera iteración se inserta si y sólo si el valor de una variable local nos que indica si hubo éxito al insertar la traza de la petición. En caso afirmativo realiza un *update* y continúa con la iteración en los elementos para insertar los datos de la respuesta en la tabla DATOS.

Con la definición de este punto quedaría definida completamente la regla de la política en sentido cliente a servidor (llegada de la petición). Para el caso de la regla de respuesta, se procedería de la misma manera. En la [figura 3.16] se puede observar el flujo de acciones que se desencadenan con la respuesta que procesa DataPower. Para el caso de los logs, se utiliza la misma estructura para el código XSLT pero adaptada al formato requerido por el XML de respuesta.

3.4.4.2 Comunicación extremo a extremo: sentido salida

Para el desarrollo de este *web service,* todos los conceptos explicados anteriormente son igualmente válidos y, dado que muchos de los objetos se van a poder reutilizar, el nivel de detalle de la explicación va a ser menor. Esto es una ventaja y a la vez un defecto, ya que es posible que al modificar un objeto determinado, los que posean dependencias del mismo se vean afectados, provocando incluso problemas de configuración si no está bien estructurado.

a) Definición del WSDL

La definición del WSDL del servicio de salida se obtiene del mismo repositorio público que el anterior. Representa una llamada a un sistema externo situado en la Empresa B que proporciona información de un determinado cliente dado un ID.

Las técnicas aplicadas para la definición del anterior WSDL son igualmente aplicables en esta ocasión. Por lo tanto, no se volverán a explicar por ser redundantes. Señalar que el fichero WSDL se almacena en DataPower, en el directorio *local:///* bajo el dominio "desarrollo" y así no depender de una conexión externa para cargarlo.

En la siguiente tabla se indica la estructura de datos para un WS en sentido salida de la Empresa A. Para este caso el número de parámetros devuelto es mucho mayor y tiene una estructura más compleja.

ENTRADA		
Nombre	**Composición**	**Tipo de dato**
Id		[A-Z]{2}-\d{5}
SALIDA		
Nombre	**Composición de dato complejo**	**Tipo de dato**
customer	person, companyAddress, id	
person	id, firstName, lastName, address*, age	
id		[A-Z]{2}-\d{5}
firstName		String
lastName		String
address	street, city, zipCode, country	
street		String
city		String
zipCode		String
country		String
age		Integer
companyAddress	street, city, zipCode, country, companyName	
companyName		String

[Tabla 3.7]: estructura del WSDL para el ejemplo de comunicación en sentido salida

b) Creación del objeto Web Service Proxy

Se crea otro objeto de tipo *Web Service Proxy* únicamente para este servicio y así tener los distintos desarrollos fácilmente diferenciados. No es recomendable mezclar la definición de varios WSDL de servicios web que tengan distinto sentido en la comunicación bajo el mismo *Web Service Proxy*: muchos objetos que se comparten entre ambos pueden crear problemas en la configuración de la conexión punto a punto, sobre todo para los objetos relacionados con el *XML Manager* (objeto relacionado con la configuración de cabeceras a nivel de protocolo para seguridad) y el *SSL Proxy*.

La configuración a implementar es similar a la anterior, eso sí, con algún pequeño cambio por ser en sentido salida de la Empresa A. Básicamente, en este caso, en vez de definir un objeto con una visibilidad hacia el exterior, se realiza con una visibilidad hacia la red local y con destino la red externa.

Dado que no se realiza control de SLA, por no tener sentido su aplicación en este tipo de comunicación, la opción *SLA Enforcement Mode* debe ser *Allow*, para tratar siempre todas las peticiones que se reciban.

c) Definición del origen (Local Endpoint) – HTTP

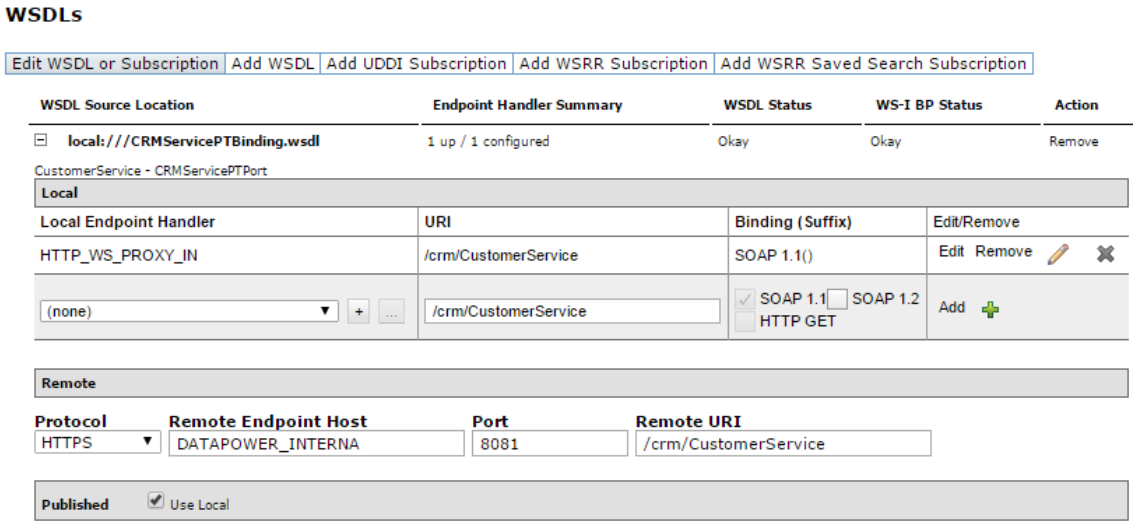
Este servicio de salida de la Empresa A tiene como origen una conexión desde nuestra red local o segura. La misma se realiza mediante protocolo HTTP, puesto que es una red de confianza. Las distintas aplicaciones llamarán entonces a una dirección HTTP con la IP de DataPower destina a uso interno. DataPower es el encargado de realizar la llamada al destino y de recopilar sus datos. Dada la configuración de la red utilizada, no sería posible la conexión directa de una aplicación de la Empresa A con las de la Empresa B sin pasar por DataPower.

Para ello se debe crear un objeto *HTTP Front Side Handler* en la definición del *Local Endpoint Handler* u origen del servicio web. La IP en la que se debe configurar es la definida en los *alias* como "DATAPOWER_INTERNA". Al ser esta IP solamente accesible desde la red segura, evitamos problemas de acceso de clientes no confiables desde el exterior. Para esta conexión, al no ser declarada bajo un protocolo seguro, no hay que definir objetos criptográficos. Un modo adicional para implementar seguridad es el de utilizar listas de acceso al servicio mediante un listado de IPs permitidas. Se indican dentro de la creación del objeto HTTP pero para nosotros no va a ser necesario.

d) Definición del destino (Remote Endpoint) – HTTPs

Al definir una salida del servicio web fuera de la empresa, es necesario definir un protocolo de comunicación segura. Por ello la plataforma destino debe prepararse para el tratamiento de este tipo de conexiones, implementando también seguridad en su comunicación. Es competencia de los sistemas definir acuerdos de interfaz para resolver este tipo de conexiones. Para definir el destino en DataPower, sólo es necesario definir en el *Web Service Proxy* la parte del *Endpoint* como HTTPs, además del *host*, el puerto y la URI del destino.

Puesto que se ha definido la conexión saliente como HTTPs, hay que definir los pertinentes objetos criptográficos. Se definen en "*Proxy Settings> SSL Proxy"*. La definición de los mismos es muy similar a la realizada para el *Web Service Proxy* de entrada. Se debe configurar un certificado de identificación para presentarnos como la Empresa A y otro de validación para aceptar conexiones de la Empresa B, esto es, definir el *handshake* como el caso anterior por tener los mismos intermediarios. El *Crypto Profile* se define dentro del *SSL Proxy*. Sin embargo, esta vez el *SSL Direction* se debe definir como *Forward,* DataPower actúa como un cliente de la comunicación. Importante este punto ya que si no, no se estará definiendo correctamente la conexión SSL.



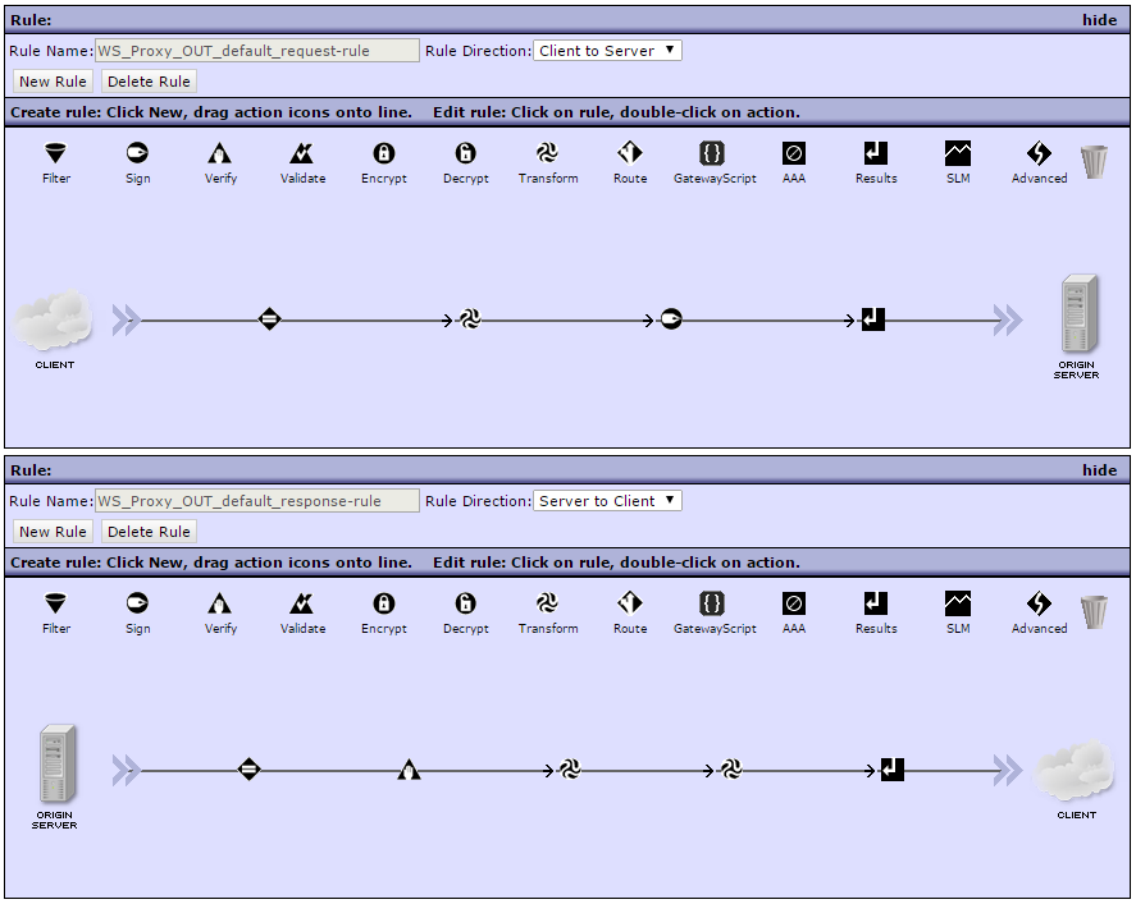
[Figura 3.18] Definición de local endpoint y remote endpoint en un servicio en sentido salida

En la [figura 3.18] se muestra la definición del punto de entrada y de salida para este servicio web. Para la salida se ha escogido la IP interna bajo un protocolo HTTPs para simular la entrada a la Empresa B. Esto significa que se necesita realizar un desarrollo adicional, lo que se va a llamar como un "simulador", para representar cierta funcionalidad

de la empresa externa a la que se llama y que es necesaria para las pruebas. Su explicación se detalla en un apartado posterior destinado a la definición de las pruebas.

e) Reglas y tratamiento de transacciones

Las reglas son análogas al caso anterior: dos reglas en distintas direcciones y sin regla específica para el tratamiento de errores realizadas a nivel de *Web Service Proxy.* Existe una diferencia primordial, no existe tratamiento de SLA o de acuerdo de nivel de servicio. Esto es debido principalmente a que este tipo de controles se implementa en los servicios web en sentido entrada y no en sentido salida.



[Figura 3.19] Reglas para Comunicación extremo a extremo de salida de DataPower

El resumen de las acciones de cada una de las reglas, para petición y respuesta, se muestra en el diagrama de la [figura 3.19]. Para la regla relativa a la petición, al recibir una de ellas, se inserta en los logs su inicio, en las tablas TRAZAS y DATOS de la misma manera que se realizó anteriormente, y se firma como la Empresa A mediante la clave privada pertinente para enviarlo a destino. Una vez recibida la respuesta, en su regla de tratamiento, se verifica la firma de la Empresa B mediante su certificado público y, en caso afirmativo, se continúa insertando los datos en los logs y devolviendo la información al peticionario, en nuestro ejemplo una aplicación de nuestra red de confianza.

f) Gestión de la firma de las transacciones

La definición de la firma y validación para este caso es muy similar al caso anterior. En la regla de la petición, al ser un servicio web en sentido salida, se realiza la firma con las credenciales de la Empresa A.

En el sentido inverso de la comunicación es necesario validar que la información que nos llega viene correctamente firmada por la Empresa B. Si es así, se elimina dicha información de la cabecera SOAP mediante una transformación y se envía a destino con los datos requeridos.

g) Trazas de la transacción en Base de Datos

El sistema de logs en base de datos se basa en el mismo concepto que anteriormente: ir recorriendo los distintos nodos de la petición y respuesta XML utilizando un conjunto definido de plantillas o *templates*. Para ello se añaden las nuevas plantillas a los documentos realizados anteriormente, reutilizando la misma estructura de fichero para ambos servicios.

Para este caso particular en la respuesta se debe recorrer un mayor número de nodos, tal y como se ha visto en su definición en la [tabla 3.7]. Esto implica un mayor número de plantillas a añadir, y no necesariamente un aumento de tiempo y recursos en su ejecución. Todos estos datos se añaden a la tabla DATOS con el nombre del campo y su valor correspondiente, y pueden relacionarse con la petición a través del número de transacción.

A modo de resumen general, comentar que conceptualmente es muy similar al anterior, reutilizando objetos y ejecutando prácticamente las mismas acciones, eso sí, sin incluir medición de servicios.

3.5 Transferencia de ficheros

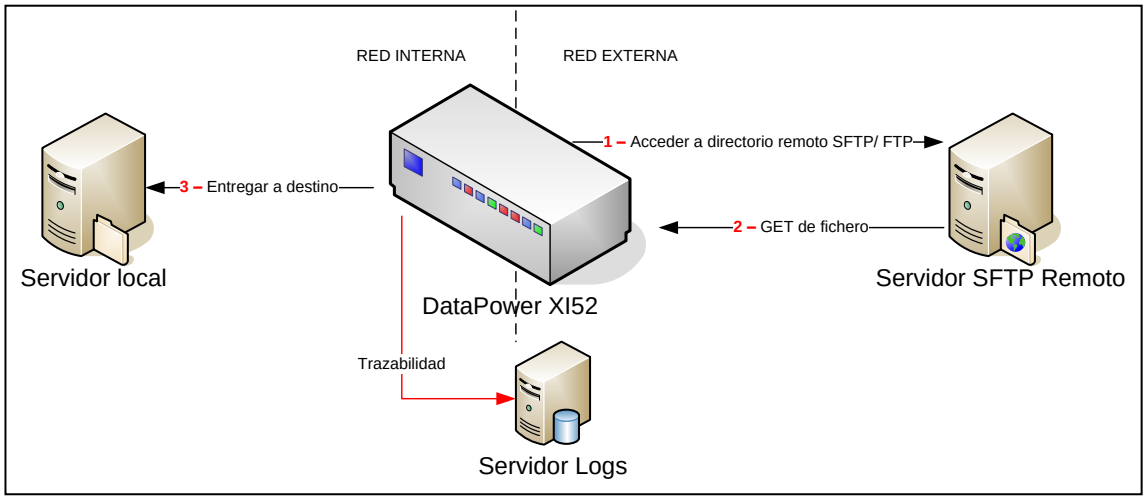
A continuación se describe la arquitectura, diseño e implementación del escenario "*Enterprise Security Gateway*" que se podría utilizar para una comunicación entre dos empresas simuladas en el que DataPower juega el rol de *Gateway* para gestionar las credenciales de una comunicación mediante un protocolo de transferencia de ficheros.

3.5.1 Arquitectura

DataPower permite configurar su comportamiento como *Gateway* con multitud de protocolos de entrada y salida. Además, no necesariamente tiene porqué ser el mismo a cada extremo; esto nos permite realizar una transformación interna, de manera transparente para los sistemas conectados, de los protocolos de comunicación.

En el ejemplo anterior, para el caso de la comunicación extremo a extremo, se realizaba un mapeo desde un protocolo HTTP a otro HTTPs y viceversa, dependiendo del sentido de la comunicación. Para la arquitectura propuesta en este apartado, se va a utilizar

otro tipo de protocolo, FTP y SFTP[85], con el objetivo de definir otro ámbito de trabajo distinto. Podríamos englobar este ejemplo en la arquitectura de DataPower denominada anteriormente como ***Enterprise Security Gateway***: su utilización va a proporcionar seguridad en las comunicaciones entre las empresas por ser DataPower el encargado de gestionar las credenciales.



[Figura 3.20]: Esquema de arquitectura de transferencia de ficheros

Para el ejemplo de este escenario en particular, se parte de la suposición de que la empresa externa con la que nos estamos comunicando utiliza un fichero para ofrecernos cierto tipo de información. En nuestro caso práctico lo utilizaremos para obtener un fichero concreto de la Empresa B para utilizarlo en la Empresa A. La arquitectura utilizada se puede ver en la [figura 3.20].

Para conseguir aplicar esta estructura se debe implementar que DataPower controle el acceso al sistema destino externo, para después obtener el fichero y guardarlo en el servidor local. De este modo, no se crea un servicio con un punto de entrada o salida como tal, sino que se debe realizar un proceso autónomo que pueda activarlo, un temporizador. Por lo tanto, para acceder a los archivos externos se utiliza un sistema de obtención de ficheros mediante temporizadores, también llamado comúnmente *poller*. DataPower controla el intervalo de ejecución de este *poller* y el tratamiento de los ficheros procesados: sin acción, renombrado o borrado. DataPower se convierte entonces en una entidad independiente que ofrece un servicio seguro de manera transparente a la Empresa A.

La elección de la arquitectura anterior tiene como finalidad que DataPower sea el único encargado de conocer y gestionar las contraseñas de los sistemas a comunicar, siendo invisible para los destinos estos datos. Se dota así a la comunicación de una mayor seguridad siendo DataPower el almacén de las contraseñas. Para que sea una aplicación independiente, como ya se ha comentado, se habilita la obtención mediante un temporizador o *poller*.

Como en el caso anterior, DataPower ofrece procedimientos para poder ver los *logs* o trazas de ejecución a través la consola de gestión. Por este motivo, también es necesario realizar un desarrollo adicional para poder describir un sistema de trazabilidad alternativo al existente por defecto. Más concretamente, se utilizará la misma base de datos para guardar

[85] **SFTP**: *Secure File Transfer Protocol.* Protocolo que define el intercambio de ficheros bajo una comunicación punto a punto implementando SSL o TLS.

las actuaciones que se realicen en los ficheros SFTP, no incluyendo tratamiento de errores como ocurre con el caso anterior. Los datos de ejecución también se podrán detectar en los ficheros que recogen los eventos de sistema *syslog*.

3.5.2 Requisitos

Como se intuye por el modelo estructural presentado anteriormente, para esta arquitectura de aplicación se requiere el uso de un par de servidores de FTP/SFTP. El único requisito principal que tienen estos servidores es que uno de ellos tiene que estar fuera de la red de confianza y el otro dentro de la misma. El servidor de FTP local, el utilizado para recibir el fichero, se va a configurar en el servidor de apoyo Linux. Al realizar esto se reutilizan recursos y no es necesario depender de otro servidor adicional. Para la comunicación se va a seguir utilizando SFTP en la medida de lo posible. La característica más importante de todo este desarrollo es que DataPower sea el único que gestione las conexiones seguras entre los sistemas involucrados, ya sea mediante usuario y contraseña o mediante claves privadas.

Se debe crear el mencionado **temporizador** o *poller* que permita ejecutar cada cierto tiempo la petición de obtención de fichero al sistema externo para después poder entregarlo a nuestro destino en local. No sería del todo necesario el tratamiento de los ficheros en el origen una vez leídos, es decir, referido a un cambio de nombre o moverlos a una carpeta una vez procesados. La implementación de este aspecto no sería muy complicada ya que es una opción que proporciona DataPower y que siempre se puede añadir una vez esté creado el servicio. Este componente es un requisito específico de este escenario y no reutiliza componentes del resto.

De cara a tener un seguimiento de las ejecuciones, se debe implantar una gestión de logs en **base de datos** similar a los casos que se han mencionado en el apartado 3.4.2. Dicha funcionalidad debe hacer uso de la base de datos en el mismo formato que los anteriores de cara a reutilizar recursos.

A modo de recopilatorio estos serían los requisitos más importantes:

- Gestión de conexiones seguras entre servidores origen y destino: almacén de credenciales.
- Montaje de servidores FTP/ SFTP: se debe implementar un servidor de ficheros de cara a guardar la entrada de datos del servicio web.
- Implementación del proceso temporizador o *Poller*: acceso de DataPower al servidor de ficheros destino.
- Gestión de logs: trazas para comprobar las ejecuciones, requisito que puede ser reutilizado en de otros componentes desarrollados.

3.5.3 Diseño

Del mismo modo que en el caso anterior, se diferencia entre el diseño de las aplicaciones de apoyo, ajenas a DataPower, y las del propio aplicativo.

3.5.3.1 Diseño de aplicaciones de apoyo

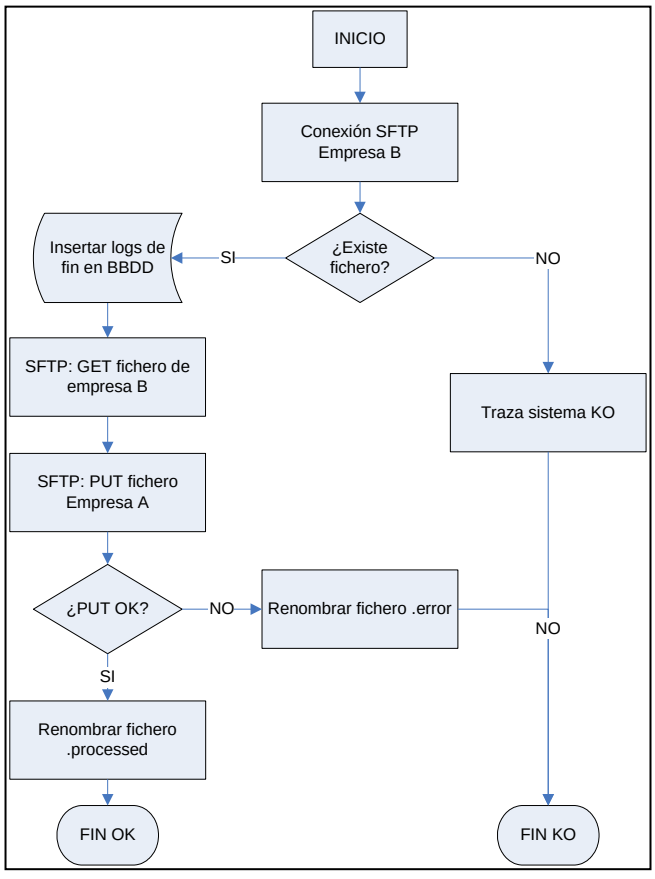
El diseño de las aplicaciones de apoyo en caso de funcionamiento como un Gateway para la transferencia de ficheros es igual a la expuesta anteriormente. Esto es, al realizar este tipo de estructura funcional sobre la misma máquina de DataPower, todo el diseño y arquitectura de las aplicaciones de apoyo es totalmente válida, por lo que se puede reutilizar.

A modo de resumen deben cumplirse los mismos requisitos:

- Diseño de la red local
- Diseño de certificados para la comunicación segura
- Diseño de la base de datos para guardado de trazas
- Diseño de trazas de sistema

3.5.3.2 Diseño de DataPower

El diagrama de estado es el que se muestra en la [figura 3.21]. El punto de comienzo lo marca un temporizador, que inicia el flujo correspondiente. Una vez realizada la conexión SFTP o FTP, si el fichero en el servidor origen existe, se genera un log en base de datos, y se realiza la copia a destino vía SFTP/FTP; dependiendo del resultado de la misma, se renombra el fichero origen con el sufijo *".processed"* o *".error",* en el caso de que haya terminado OK o KO respectivamente.



[Figura 3.21]: Diagrama de estados para la transferencia de ficheros

3.5.4 Implementación del servicio en DataPower

La implementación de este escenario en DataPower consta de los siguientes pasos diferenciados:

- Creación del objeto *Multi Protocol Gateway* (MPG) en DataPower
- Definición del origen del MPG
- Definición del destino del MPG
- Reglas y tratamiento de las transacciones
- Trazas de ejecución en Base de Datos y en un servidor externo mediante Syslog
- Creación de puente para comunicaciones entre clientes de la empresa A hacia la Empresa B

a) Creación del objeto Multi-Protocol Gateway

Para la definición de un proceso que realiza la transferencia de ficheros se debe utilizar un objeto llamado *Multi-Protocol Gateway*[86]. Como su propio nombre indica, se trata del mismo proceso *Gateway* utilizado anteriormente pero abarcando más posibilidades como protocolos de E/S. Ya no se centra en la comunicación SOAP/ XML sobre HTTP/S entre origen y destino, sino que añade capacidades de lectura de colas MQ, TIBCO EMS, protocolos FTP, etc.

Para acceder al mismo, se debe pulsar desde el panel de control el icono:  o acceder desde el menú "*Objects>Service Configuration>Multi-Protocol Gateway*". Es necesario definir los siguientes objetos para configurarlo:

- *Front Side Protocol*: relativo al protocolo de entrada. Dentro del mismo se deben definir los objetos relativos a las credenciales en caso de ser un protocolo con seguridad.
- *XML Manager:* gestión de la recopilación y de la caché para las hojas de estilo.
- *Agent User:* objeto contenido en el *XML Manager.* Se utiliza para modificar el detalle de las conexiones de red de salida del aplicativo.
- *Policy:* reglas de tratamiento de las transacciones.
- *SSL Crypto Profile:* Objeto de credenciales referido al *backend.*

b) Definición del origen – FTP

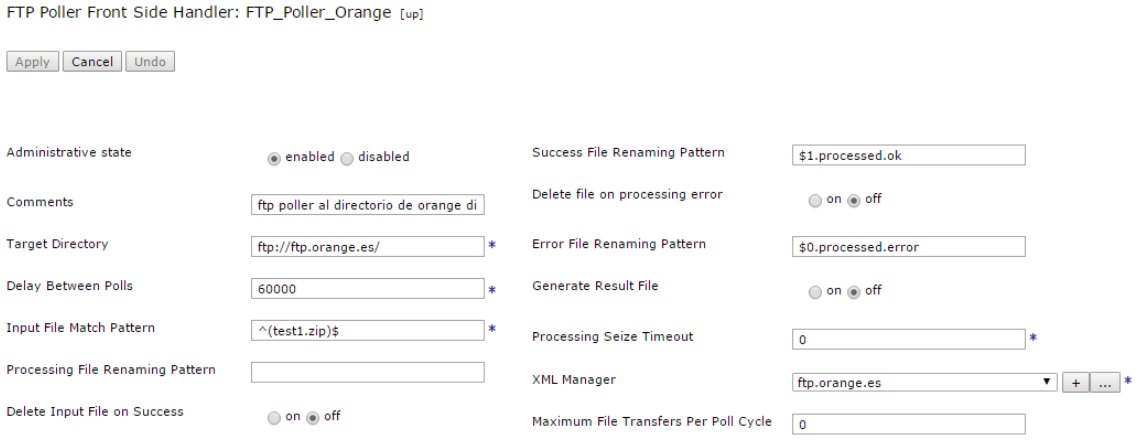
Como se ha indicado en el diseño, el origen de los datos es un acceso remoto a un servidor de ficheros. El acceso al servidor de ficheros externo se realiza mediante FTP. Esto no es la práctica habitual, pero a efectos de demostrar cómo se realizaría un tipo de conexión basado en transferencia de ficheros, sería suficiente. Para el destino se implementa SFTP.

Por lo tanto, como entrada o *Front Side Protocol* se elige *FTP Poller Front Side Handler*. Este objeto representa una conexión activa hacia un servidor FTP para localizar un fichero. Se pueden definir además varios protocolos simultáneamente, lo que conlleva que se tengan varias entradas al servicio. Para nuestro caso sólo es necesaria una entrada.

En el objeto *FTP Poller Front Side Handler*[87] se configura principalmente el destino, el intervalo de recuperación y el patrón de los ficheros. Para el destino o *Target Directory*, se

[86] La definición de los objetos *Multi-Protocol Gateway* de DataPower se describen en el capítulo 6 de la referencia [46]
[87] Para más información acerca de la configuración de los objetos de obtención de ficheros mediante FTP/ SFTP ver el *RedBook* de IBM de la referencia [54]

debe indicar el protocolo a utilizar así como indicar también la ruta absoluta. Para el patrón de ficheros a buscar se configura el valor "*^(test1.zip)$*" para que busque el fichero con nombre *test1.zip* y se renombre posteriormente, indicado con los paréntesis.



[Figura 3.22] Configuración de FTP Poller Front Side Handler

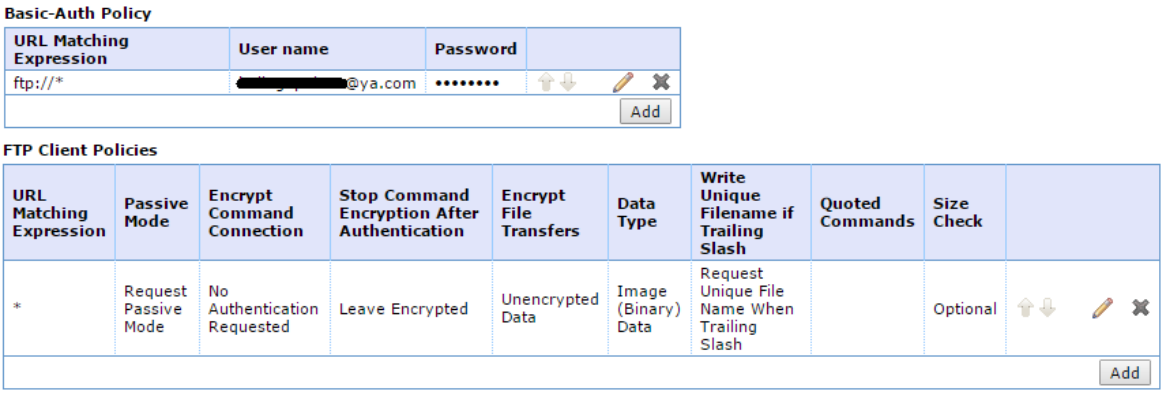
Se configura el renombrado de ficheros para que, una vez leído, se marque en el servidor como "*.processed.ok*" o "*.processed.error"* si existiera error. En caso de no habilitar el renombrado, en la siguiente ejecución el fichero se volvería a procesar de nuevo. Si el fichero no existe, DataPower generaría una traza de sistema de tipo *notify* indicando que no existe y parando la ejecución. La traza podría verse en el fichero de logs de eventos de DataPower alojado en la máquina UBUNTU en *"/var/log/datapower.log"* ya que se recogen desde ese grado de notificaciones hasta el más crítico. La notificación mostrada sería similar a:

```
        Apr 24 16:49:38 DP_SERGIO [0x80e000d2][file-poller][notice] source-ftp-
poller(FTP_Poller_Orange): trans(26671): No input found for URL ftp://ftp.orange.es/'
```

Dado que el acceso al FTP requiere de autenticación simple en el servidor, hay que indicar el usuario y la contraseña en las cabeceras de la conexión. Para definirlo se utiliza el objeto *User Agent*, contenido dentro del *XML Manager*. Con éste podemos configurar a bajo nivel todo lo relacionado con la conexión con el destino, desde inyectar cabeceras nuevas, hasta política de *SSL Proxy* por tipo de dato recibido, pasando por las políticas de conexión de FTP y SFTP.

Para nuestro desarrollo se utilizan las características "*Basic Auth-Policy*" y *"FTP Client Policies"*. En el primero se define el usuario y contraseña de la conexión FTP; se va a indicar un único registro de usuario para cualquier conexión FTP realizada por el *Poller*. En el segundo, la política de tratamiento para la transmisión de datos FTP. Es importante configurar el tipo de dato a transmitir. Como se va a transmitir un fichero en formato comprimido se debe seleccionar binario. El resto de parámetros se dejan por defecto.

El detalle de la configuración se muestra en la [figura 3.23]. Se utiliza un servidor FTP del dominio *@ya.com* a modo de ejemplo de servidor ajeno a la red local. Como se ha comentado anteriormente, para este desarrollo es DataPower el encargado de gestionar el usuario y contraseña de conexión al destino.



[Figura 3.23] Configuración del Agent User para definir la conexión de red de un FTP Poller

c) Definición del destino – SFTP

El destino se configura en la pestaña General del *Multi-Protocol Gateway*. Dicho destino identifica unívocamente el protocolo y la ruta completa, incluida el nombre del fichero. En nuestra solución el protocolo a utilizar va a ser SFTP y el destino la máquina UBUNTU de la red local. Se debe definir la ruta completa, eso significa para este ejemplo que deba definirse el nombre del fichero que se va a almacenar en el destino:

```
sftp://UBUNTU/work/test1.zip
```

El servidor destino utiliza un protocolo seguro con autenticación. La configuración de DataPower para habilitar este tipo de conexión se encuentra dentro el *XML Manager* y el *User Agent* relativos al destino o *backEnd*. Estos objetos van a definir el tratamiento a nivel de cabeceras, incluyendo en las mismas los campos de autenticación necesarios para el destino. Para la definición de los servicios web de comunicación punto a punto no hizo falta configurar los objetos por no tener estas características. Para esta ocasión, el servidor SFTP requiere una autenticación basada en usuario y contraseña, por lo que es necesario, como en el caso de la definición del origen, configurar esta clase de objetos. El origen del servicio se ha configurado en el apartado anterior como un acceso *poller* a un servidor de ficheros mediante FTP; ergo esta conexión se puede considerar como otro destino adicional. Por ello es necesario configurar una vez para cada servidor de ficheros, FTP de origen y SFTP de destino, los objetos del *XML Manager*.

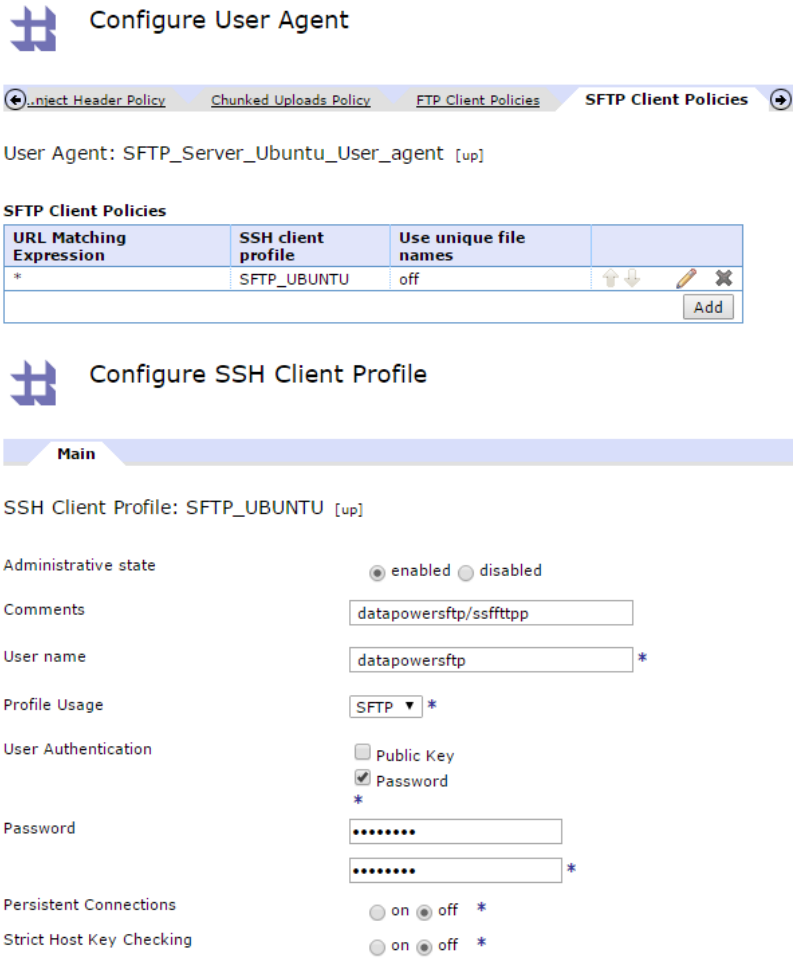
Relativo al *User Agent* de destino, para el mismo se debe configurar las características de la pestaña *SFTP Client Policies*. Dentro se define, para un determinado SFTP, un objeto de conexión *SSH Client* Profile; contiene básicamente el usuario y contraseña. Dado que es una conexión sobre SSH[88], se podría de igual manera identificar el usuario mediante su clave privada. La opción está dentro de este mismo objeto, pero no se va a utilizar en este caso por la configuración del servidor UBUNTU. Además de lo anterior, se debe seleccionar que las conexiones no sean persistentes para no permitir que las nuevas peticiones SSH eviten la autenticación si tienen una sesión previa. La transferencia consta de un único fichero, por lo que no compete el activarlo. Su detalle se muestra en la [figura 3.24].

A modo de resumen, esta es la lista de objetos a configurar para el destino:

XML Manager **>** *User Agent* **>** *SFTP Client Policies* **>** *SSH Client Profile*

[88] **SSH:** *Secure Shell.* Telnet para realizar una conexión a una máquina remota utilizando el protocolo seguro SSL.

En el caso en el que el origen de los datos (el *Poller* de entrada) no hubiera sido una conexión FTP sino SFTP, la configuración a realizar sería la equivalente a la realizada en este apartado. Se debe definir dentro del *XML Manager* un objeto *User Agent* que contenga una política de conexión a SFTP para un determinado perfil de SSH. El perfil definiría el tipo de autenticación, mediante certificado o usuario y contraseña.



[Figura 3.24] Definición de User Agent, SFTP Policies y SSH Client Profile para una conexión SFTP

d) Reglas y tratamiento de las transacciones

Para este ejemplo, las reglas a implementar en la *Policy* del *Multi-Protocol Gateway* son bastante más sencillas que las anteriores. En parte porque no se realiza ni firma y ni validación de las transacciones, ni tampoco verificación de SLA. Por lo tanto, sólo se incluirá una regla de tipo *Transformation* para insertar una petición en base de datos. Con esta petición se busca indicar, a modo informativo, que se ha realizado una petición de tipo PUT para el fichero.

e) Trazas de ejecución: base de datos y *syslog*

El XSLT a utilizar es mucho más sencillo que el anterior, puesto que se va a insertar sólo en la tabla TRAZAS en un único paso, no realizando la inserción a dos tiempos dada una

petición y una respuesta. Tampoco se insertarán los datos en la tabla DATOS. Se busca sencillez y funcionalidad. En caso de errores en la ejecución pueden verse fácilmente en el registro de logs del sistema (*syslog*).

El XSLT se compone en este caso sólo de la definición de los datos y de la ejecución de la función *dp:sql-execute* de DataPower para insertar el registro dentro del *template* principal definido con el *tag "/"*. No se itera sobre elementos XML puesto que aquí no existe entrada de datos propiamente dicha. El protocolo utilizado no es SOAP, por lo que no se define la entrada como un estándar XML. Aunque realmente lo que se está modificando en la transformación es un XML vacío de entrada, conceptualmente consiste en ejecutar determinadas sentencias propias de DataPower que nos permiten insertar en base de datos.

f) Enlace puente desde red local a servidores de ficheros

Como se ha comentado anteriormente, DataPower va a ser el encargado de gestionar las contraseñas, lo que implica la gestión del acceso a los destinos definidos en el *Poller* y en el *BackEnd.* Existen limitaciones a este modelo, como por ejemplo el permitir a los usuarios de la Empresa A acceder al servidor de ficheros de la Empresa B si no conocen su contraseña. Además, tampoco podrían acceder directamente realizando un SFTP/FTP desde su máquina dado que el *Gateway* de salida de la red local (en nuestro caso el *router*) no le permite acceder a redes externas si no es a través de DataPower. Como solución a este problema se definen conexiones "puente" entre una IP y puerto de DataPower en la red local y un destino de la red externa. Con esto, el usuario sólo necesitaría acreditarse en la interfaz creada en DataPower a tal efecto para acceder al destino. Lo haría realizando una conexión SFTP con la dirección IP de la máquina de DataPower, que serviría como espejo de la de destino. Seguiría sin conocer el usuario y contraseña originales y además le permitiría acceder a los sistemas externos por pasar la validación del *Gateway*.

Para ello se crean dos objetos de tipo *Multi-Protocol Gateway.* Uno de ellos realizará la conexión desde la red local hasta el destino del objeto *Poller*, mientras que el otro será hacia el servidor montado en la red local. Ambos tienen conceptos similares en cuanto a implementación, únicamente cambia el destino de los mismos, lo que implica que la autenticación sea distinta.

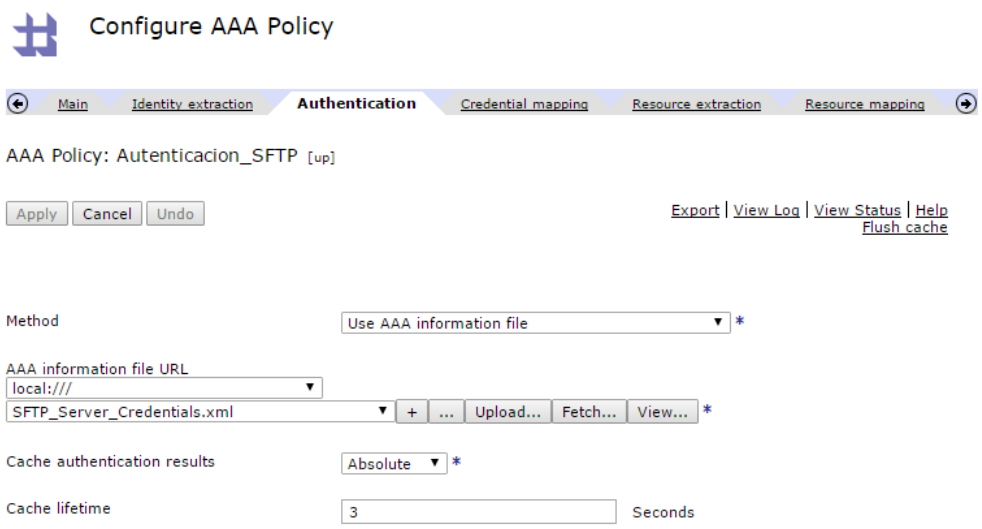
Para el servidor en red externa se ha elegido un destino público que ofrece un servidor de ficheros SFTP. El mismo sólo permite el acceso y no la modificación de los ficheros, por ese motivo se descartó para realizar la implementación. Sí que tiene sentido para mostrar cómo se realiza una conexión directa desde la red local hasta el servidor SFTP destino.

Dicho enlace debe definir como origen un *SFTP Server Front Side Handler.* Con ello se ofrece en una IP y puerto un servidor SFTP. La IP elegida es la correspodiente al ethernet *eth0* o configurada en el alias como "DATAPOWER_INTERNA". En este objeto además hay que configurar la autenticación con nuestro servidor o *AAA Policy*. La configuración del acceso se puede realizar con un servidor LDAP, el cual nos podría servir también para autenticar al usuario en DataPower, como ya se vió antes, utilizandolo como gestor de identidades. Se gana en dinamismo pero se pierde en coherencia, ya que uno de los objetivos que buscamos en nuestro desarrollo es aplicar ese rol a DataPower para mostrar sus posibilidades. Si se diera el caso de que en la red de la empresa en la que se instale DataPower ya existiera un servidor LDAP, sí sería recomendable utilizarlo. En nuestro ejemplo no se va a realizar así. Por lo tanto, se simplifica la implementación haciendo de DataPower el único encargado de gestionar los usuarios y contraseñas para los accesos a los WS de "puente". Se implementa mediante un fichero XML con las credenciales instalado dentro de DataPower.

Una configuración básica del fichero de credenciales[89] podría ser la siguiente:

```
<aaa:AAAInfo xmlns:dpfunc="http://www.datapower.com/extensions/functions"
xmlns:aaa="http://www.datapower.com/AAAInfo">
    <aaa:FormatVersion>1</aaa:FormatVersion>
    <aaa:Filename>local:///SFTP_Server_Credentials.xml</aaa:Filename>
    <aaa:Authenticate>
        <aaa:Username>user</aaa:Username>
        <aaa:Password>user01</aaa:Password>
        <aaa:OutputCredential>Credencial_de_SFTP</aaa:OutputCredential>
    </aaa:Authenticate>
</aaa:AAAInfo>
```

De este modo indicado, se configuraría de manera rápida un enlace entre la red local y el destino utilizando un usuario y password definidos en DataPower. En la [figura 3.25] se puede observar la configuración de autenticación dentro del objeto *SFTP Server Front Side Handler* utilizando un fichero de credenciales como el anterior.



[Figura 3.25] AAA Policy para un servidor SFTP en DataPower

Respecto al destino es similar a los anteriores, configurar dentro del *XML Manager* un objeto de tipo *User Agent* que incluya la autenticación SSH en los paquetes que se envíen a destino.

Con todo lo explicado en este apartado, quedaría completamente configurado el enlace de apoyo desde un cliente de la Empresa A hasta el servidor de ficheros de la Empresa B.

[89] La estructura general de los ficheros de credenciales de DataPower se define en el capítulo 16 de la referencia [46]

3.6 ESB – Enterprise Service Bus

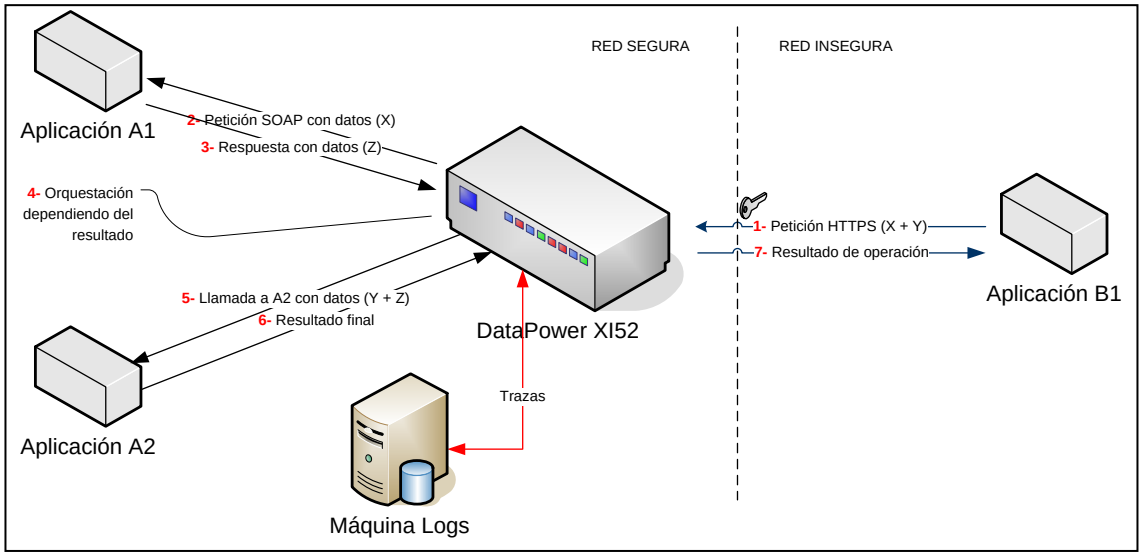
A continuación se describe la arquitectura, diseño e implementación del escenario "*Enterprise Service Bus*" que se podría utilizar para una comunicación entre dos empresas simuladas en el que DataPower juega el rol de *Gateway* ESB.

3.6.1 Arquitectura

El último caso de uso a implementar es el de funcionamiento como **ESB**, en el que se va a utilizar la capacidad de procesamiento y de tratamiento de peticiones de DataPower para incluir una lógica de negocio adicional. Conceptualmente es muy similar a lo que puede ser la comunicación extremo a extremo que se mencionó al principio, ya que a efectos prácticos, para los sistemas externos, DataPower se comporta al final como una caja negra, independientemente del número de llamadas a otras aplicaciones o la lógica adicional que se realice de manera interna.

En la [figura 3.26] se muestra un ejemplo de esta arquitectura con varias peticiones a sistemas externos. Como en el caso de comunicación de extremo a extremo, se opta por utilizar XML formateado mediante la definición SOAP y enviarlo utilizando los protocolos HTTP/ HTTPs. Las motivaciones para su elección son claras: compatibilidad, capacidad de procesamiento y escalabilidad.

El objetivo buscado para este desarrollo es dejar a DataPower la gestión de las llamadas a los servicios web, así como su tratamiento de datos para realizar otra llamada más compleja. El tratamiento de estas llamadas o ejecuciones adicionales, así como la gestión de los datos recibidos, recae sobre DataPower, permitiendo liberar de carga al resto de sistemas implicados en la comunicación.



[Figura 3.26]: Esquema de arquitectura ESB

El flujo de datos propuesto para esta arquitectura comienza cuando recibimos una nueva petición SOAP desde la Empresa B hacia nuestro DataPower. En esa petición se

incluyen los datos "(X+Y)". DataPower se encarga de gestionar la petición a una aplicación interna A1 utilizando los datos "(X)" para obtener datos intermedios "(Z)", para después realizar otra petición hacia la aplicación A2 con una mezcla de ambos "(Y+Z)". Posteriormente se devuelve el resultado de la operación al peticionario, incluyendo algunos datos de operaciones intermedias, "(Z)".

Respecto al ejemplo ilustrativo sobre el que se va a trabajar para esta arquitectura, se va a implementar un **servicio de entrada** que permita añadir al carro de la compra determinados productos para un cliente. Se ha buscado un ejemplo sencillo y gráfico en el que exista intercambio de datos. Nuestra Empresa A actúa como un maestro de datos, organizando las llamadas a las distintas aplicaciones y tratando los datos de respuesta, para devolverlo a la Empresa B.

Como en los casos anteriores y motivado por el déficit de DataPower para el tratamiento de trazas de ejecución, este servicio web está apoyado por un sistema de guardado de trazas en base de datos reutilizando los recursos creados para todos los servicios del aplicativo.

3.6.2 Requisitos

Referido a la implementación de una comunicación con DataPower utilizando una estructura de funcionamiento similar a un ESB, los requisitos y el concepto son prácticamente iguales a los de la comunicación "extremo a extremo". Partiendo de esa base, se le añade una pequeña lógica de negocio interna que sea la encargada de gestionar los datos y diversas llamadas adicionales. Toda esta lógica de negocio hay que programarla utilizando el lenguaje de transformación de XML llamado XSLT. Un apoyo importante para la programación es el utilizar las funciones extendidas propietarias de DataPower así como el uso de las variables de contexto. Esto permitirá mayor flexibilidad y velocidad en la ejecución de las transformaciones.

En la [figura 3.26] se ha mostrado el esquema deseado para esta implementación. Dado que hay una **orquestación de aplicaciones**, hay que añadir un tratamiento en el caso de que se produzca un error en la gestión de las peticiones. Esto va a conllevar a tener que realizar reglas para el tratamiento de errores derivados de las consultas a sistemas internos (*timeouts*, indisponibilidad, error en los datos enviados, etc.). Todas las reglas de tratamiento de error se van a implementar en DataPower, siendo el encargado único de la gestión del flujo de negocio.

También, de la misma manera que en la comunicación SOAP del apartado anterior, se va a implementar un sistema de logs en base de datos y las mismas medidas de seguridad en las comunicaciones.

Por lo tanto, esta es la lista de requisitos nuevos a cumplir:

- **ESB**: DataPower se comporta como un gestor de los datos necesarios en cada uno de los servicios web para proporcionar la funcionalidad necesaria de cara al sistema que nos invoca.
- **Control de errores de ejecución**: creación de reglas en DataPower para el tratamiento de errores o excepciones en la ejecución de los servicios.
-

Los siguientes requisitos son comunes con las de la implementación de una comunicación SOAP de entrada y salida:

- Credenciales de las comunicaciones.
- Blindar físicamente los servicios web HTTP utilizando distintas interfaces de red.
- Implementación de logs en base de datos.
- Implementación de logs administrativos.

El último obstáculo a salvar, común para las tres implementaciones, es la gestión de la realización de las pruebas. La simulación se realiza en el entorno de una red privada local desde la cual se tiene solamente acceso al exterior a través de DataPower; esto es, las aplicaciones internas son incapaces de realizar llamadas directamente al servicio web final si no es pasando a través de DataPower. Por lo tanto, es importante señalar que DataPower es el punto de entrada y de salida por el que tienen que transitar todas las comunicaciones de las aplicaciones de la red local. El objetivo que se consigue es el de mantener la seguridad de la red local blindando las aplicaciones internas.

3.6.3 Diseño

La fase de diseño se divide en dos etapas, una para las aplicaciones externas y otra para definir las aplicaciones de DataPower.

3.6.3.1 Diseño de aplicaciones de apoyo

Como en las dos propuestas anteriores, se van a reutilizar los componentes ya desarrollados, adaptando los desarrollos de DataPower para aprovechar todas sus características, como por ejemplo en el caso de las trazas en base de datos.

Por ello, esto no hace más que corroborar que el diseño de los componentes adicionales ha sido el correcto y son perfectamente reutilizables para multitud de desarrollos.

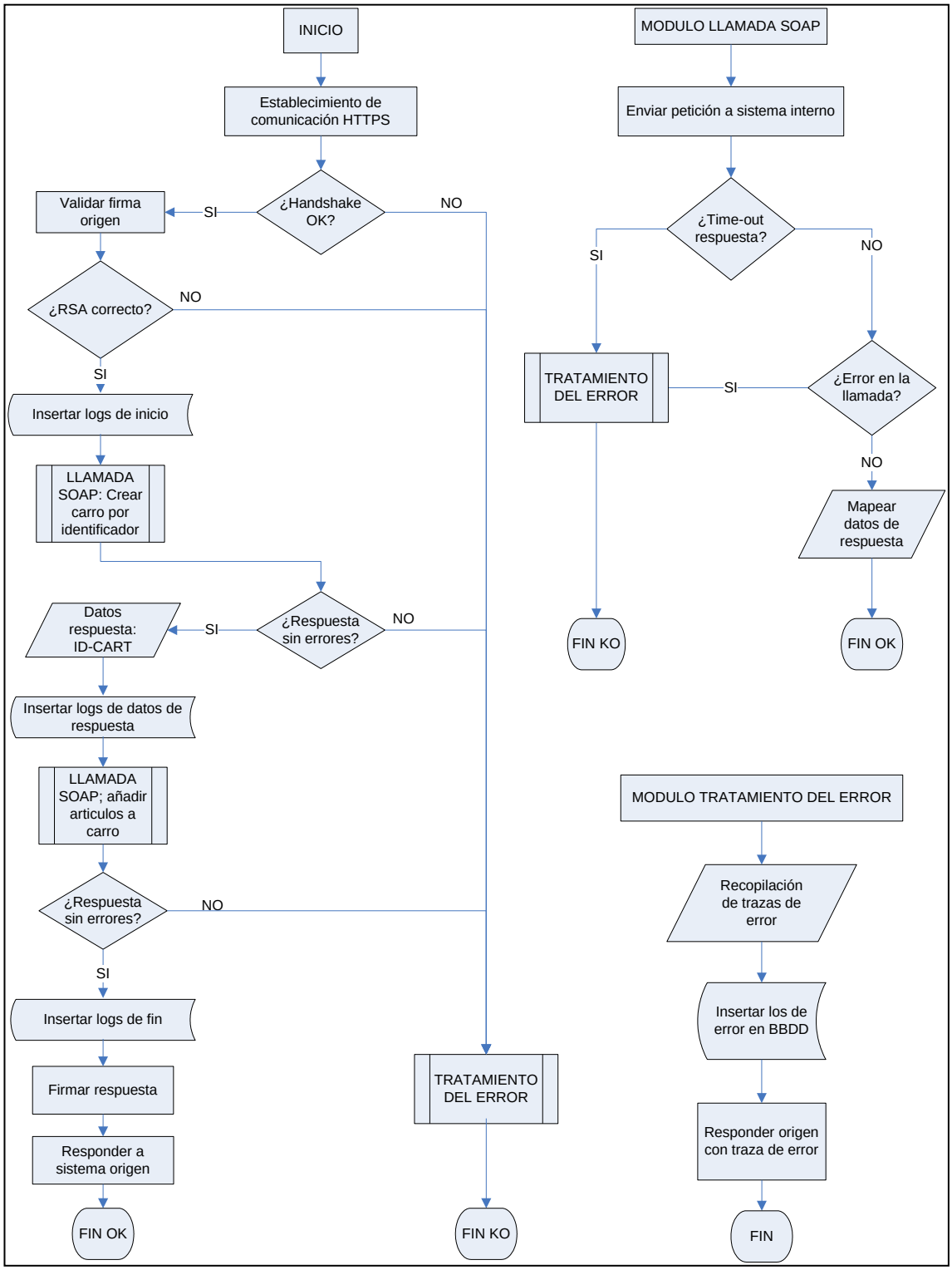
3.6.3.2 Diseño de DataPower

El evento que genera el inicio de la [figura 3.27], que representa el diagrama de estados de este servicio, es una ejecución HTTPs de entrada a DataPower. La llamada se realiza hacia la interfaz de red *eth1*.

Respecto a la funcionalidad, se proporcionará un método para añadir objetos a la lista de una compra de un determinado usuario desde la Empresa B hacia la Empresa A. Por lo tanto, la figura representa una implementación para un servicio web en sentido entrada a nuestra empresa. No va a existir otro similar de salida. El diagrama consta de un flujo principal y de dos sub-procesos reutilizables: uno es la llamada SOAP a otro servicio y otro es el tratamiento de errores del flujo.

Para este servicio web no se ofrece medición del acuerdo de nivel de servicio (SLA) que, como se ha dicho en otras ocasiones, puede implementase a posteriori como en el caso del *WS-Proxy* en sentido entrada.

El proceso de establecimiento de la comunicación y de validación de credenciales es similar al diseñado para una comunicación SOAP en sentido entrada: certificados para la comunicación TSL y validación de la firma de ambas empresas.



[Figura 3.27]: Diagrama de estados para un WS Proxy con ESB

Se va a realizar la **gestión de los errores** ocurridos a lo largo del flujo de ejecución mediante una regla específica implementada en DataPower. Sólo se ejecutará en caso de que exista error y permitirá recopilar trazas descriptivas para así realizar un análisis posterior: dentro del sub-procedimiento se van a guardar los datos del error y se va a responder al cliente con el detalle de los mismos. En el diagrama de flujo se puede observar que existen

verificaciones de errores técnicos tales como *timeouts* o similares al realizar la llamada SOAP. Aunque estas verificaciones no sean implementadas como tal en la estructura del XSLT, sí que merece la pena anotar que se van a realizar de cara a tener un control de la ejecución. Añadir que en caso de que ocurra cualquier tipo de error en el flujo, el mismo se tratará por la regla de error tal y como si se tratase de una excepción, por lo que no es necesariamente en los puntos donde se realizan comprobaciones donde pueden ocurrir los mismos.

El tratamiento de logs es similar al empleado en ocasiones anteriores, guardando en **base de datos** el dato intermedio obtenido en la primera llamada SOAP (creación del carro en este servicio web). En caso de que en la segunda llamada SOAP (añadir compra al carro) existiera un error de cualquier tipo, no se realizaría un *rollback* o marcha atrás de los datos de aplicación para la primera llamada, ya que con cada nueva iteración se va a crear un carro nuevo. Esto es una limitación derivada del simplismo con el que están tratando los servicios web. En una empresa real siempre se ofrecería un mecanismo alternativo para realizar un borrado de esos datos y tener toda la información consolidada y sincronizada en todos los sistemas implicados.

3.6.4 Implementación del servicio en DataPower

En la implementación de este WS con lógica de negocio hay que realizar los siguientes pasos:

- Definición del WSDL de la comunicación
- Creación de un objeto *Web Service Proxy* (WSP) en DataPower
- Definición del origen del WS de DataPower
- Definición del destino del WS de DataPower
- Reglas y tratamiento de las transacciones que se tramitan por el WS
- Gestión de la firma de las transacciones
- Implementación de las trazas del WS en Base de Datos

a) Definición del WSDL

El comportamiento descrito a continuación no es más que el desarrollo para un ejemplo concreto de cara a demostrar las capacidades de DataPower, y no debe tomarse como un procedimiento estandarizado.

Para el ejemplo del servicio web con características de ESB, se ha creado manualmente el WSDL por requerir un tratamiento de campos específico. Esto respalda una afirmación anterior, ya que en el caso en el que se trate de utilizar a DataPower sólo como un *proxy* en las comunicaciones, se puede importar directamente el WSDL del servicio a atacar. Para el ello se realizan dos llamadas a distintos servicios web, con datos de entrada y salida cada uno de ellos por separado. Para ello se ha partido de la definición de dos servicios ya existentes, los cuales han sido importados desde la dirección pública mencionada en párrafos anteriores. La creación de los mismos es, por consiguiente, una composición de los existentes.

En lo que respecta a la funcionalidad del servicio a crear, se define una entrada de datos, a los que podemos llamar (X+Y), de manera que con ellos se realice una petición a otro servicio web de un sistema de la red local para obtener un nuevo parámetro. En concreto, utilizando el parámetro X de la entrada, se obtiene el parámetro Z. Una vez con ese dato, se puede volver a realizar la llamada al siguiente servicio web con los parámetros

(X+Z) como entrada. Como resultado de las operaciones totales, se devuelve al llamante el dato Z obtenido en la llamada intermedia. Así se le ofrece visibilidad al peticionario del resultado de la llamada intermedia realizada por DataPower.

Todo lo anterior se resume en las siguientes tablas en las que se muestra la estructura de datos de entrada y salida para cada servicio web. El servicio llamado *addToCartByCustomerId* mostrado en la [tabla 3.8] es el que se implementa finalmente en DataPower dentro del *Web Service Proxy*; es el creado a medida y que realmente engloba a los otros dos que se realizan de manera interna.

ENTRADA	
Nombre	**Tipo de dato**
customerId	[A-Z]{2}-\d{5}
articleId	String
quantity	Integer
SALIDA	
Nombre	**Tipo de dato**
cartId?	String
error?	String
description?	String

[Tabla 3.8]: Estructura del WSDL de orquestación de DataPower addToCartByCustomerId

ENTRADA	
Nombre	**Tipo de dato**
customerId	[A-Z]{2}-\d{5}
SALIDA	
Nombre	**Tipo de dato**
cartId	String

[Tabla 3.9]: Estructura del WSDL de Creación de Carro

ENTRADA	
Nombre	**Tipo de dato**
cartId	String
articleId	String
quantity	Integer
SALIDA	
Nombre	**Tipo de dato**
addToCartResponse	*null*

[Tabla 3.10]: Estructura del WSDL de Añadir al Carro

Para el caso del WSDL del servicio *addToCartByCustomerId* de la [tabla 3.8], la respuesta contiene todos los parámetros opcionales (indicados con un '?'). Se ha elegido así puesto que, si no es capaz de recuperar el parámetro *cartId* de la llamada intermedia porque ocurra algún error, el dato a devolver al origen no existiría.

En el caso de que no exista error y se devuelva la respuesta correctamente, los campos error y descripción tampoco deberían existir. Se podrían dejar como parámetros obligatorios, pero en dicho caso se debería siempre devolver la estructura con el dato vacío. La elección de definirlos como opcionales es más dinámica y puede contemplar también esa posibilidad en la respuesta.

Para la [tabla 3.9] se define un parámetro de entrada de tipo *customerId* y un parámetro de vuelta de tipo *String* que se va a utilizar como entrada para el servicio de la [tabla 3.10] y que también se va a devolver en la salida del servicio de la [tabla 3.8].

b) Creación del objeto Web Service Proxy

La creación es muy similar a lo comentado en los casos anteriores. Se crea un *Web Service Proxy* el cual contenga como definición el WSDL específico para este servicio, creado a medida y que se ha cargado previamente en el directorio *local:///*.

c) Definición del origen (Local Endpoint) – HTTPs

El canal de entrada del servicio es la dirección de DataPower destinada a las conexiones externas, "DATAPOWER_EXTERNA" (la interfaz *eth1)*. El objeto de tipo *Front Side Handler* a definir es similar al definido anteriormente para el servicio web de la comunicación punto a punto en sentido entrada, por lo que se puede reutilizar. Por lo tanto, ambas conexiones van a compartir la misma configuración de IP, puerto y objetos criptográficos. La diferencia de cara a ejecutar uno u otro servicio radica en la URI de destino. Para este caso se define la URI: *addToCartByCustomerId*.

d) Definición de los destinos (Remote Endpoint) – HTTP

El servicio a implementar debe realizar la llamada a dos sistemas finales en vez de a uno. El modo de realizarlo es seleccionando uno de tipo "*static from WSDL"* tal y como se había definido para los anteriores e indicando como *remote endpoint* el segundo destino. El primer destino o destino intermedio se va a gestionar mediante una tarea en la regla de tratamiento de tipo *result*  .

Por lo tanto, el primer destino se indica directamente sobre la regla de tratamiento de la política de peticiones; el segundo destino se define a nivel de objeto *Web Service Proxy* como en ocasiones anteriores. Ambos destinos están accediendo a aplicaciones de la red interna, en concreto a aplicaciones *mock service* levantadas en la máquina de UBUNTU mediante la herramienta SoapUI. Como se ha visto anteriormente, con este tipo de aplicaciones se va a poder configurar la respuesta a devolver para cada ejecución.

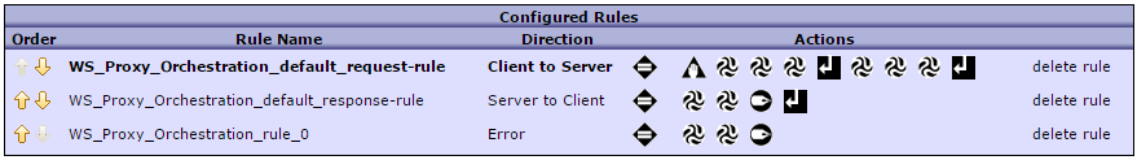
e) Reglas y tratamiento de las excepciones

Mediante una correcta definición de las reglas de las transacciones se define la funcionalidad de DataPower como un ESB. Por lo tanto este apartado va a ser un poco más complicado que el de los anteriores servicios por contener más acciones en cada una de ellas con la finalidad de implementar una pequeña lógica de negocio.

DataPower trata las transacciones para la petición y para la respuesta mediante reglas. Pueden ser sólo definidas para cada una de las direcciones, petición o respuesta, o

para ambas. Es obligatorio que siempre exista una regla para tratar las transacciones, aunque dentro no se hagan acciones adicionales. En el caso práctico que se va a exponer, además, se implementa una regla adicional para tratar los errores. Para definirla se escoge en el campo *direction*, el valor *error.*

En la [figura 3.28] se indica el resumen de las acciones para cada una de las reglas de este servicio. El primer caso representa la regla de cliente a servidor, correspondiente a una petición entrante a DataPower. El segundo representa a una regla desde servidor a cliente, es decir, la de respuesta desde DataPower al origen. Por último se define la regla de error que se ejecuta cuando se genera una excepción en cualquiera de las dos anteriores.



[Figura 3.28] Reglas para el WS Proxy con lógica de negocio

Para el caso de la **regla de entrada** se ha dividido en diez acciones que funcionalmente se comportan como lo expuesto en la fase de diseño.

1. Acción de tipo *Match.* Esta acción existe siempre para todas las reglas, la crea DataPower por defecto, y consiste en un filtrado mediante el cual, sólo si se cumplen determinadas condiciones, se va a continuar la ejecución de las acciones de la misma. Es utilizado cuando existen distintas reglas definidas para un mismo sentido, de cara a definir cuándo es utilizada cada una; por ejemplo, si se ha ejecutado una URI concreta, si viene de un origen determinado, etc. Para nuestro ejemplo no es necesario configurar ninguna acción especial y se deja con el valor por defecto para filtrar por todas las peticiones.
2. Acción de tipo *Validation* la cual consiste en una verificación de las credenciales del peticionario incluidas como firma del mensaje. Esto es, se comprueba si la firma que se recibe dentro de la petición SOAP corresponde con el certificado de la empresa en la que confiamos. Se utiliza el objeto de DataPower *Validation Credential* configurado con el valor del certificado público de la Empresa B. En caso de que no sea así o exista algún error en la firma, se rechaza generando un error.
3. Es una transformación de código XML. Con esta acción se define la inserción en la base de datos de las trazas de inicio de petición, más en concreto la correspondiente fila de la tabla TRAZAS, y la inserción de los datos de la petición, en la tabla DATOS. El código utilizado es similar al explicado para los servicios web anteriores, utilizando la recursividad, pero adaptado a la estructura del XML a procesar. Como entrada tiene el contexto INPUT, es decir, la entrada de datos a la regla, y como salida NULL, no devuelve datos transformados.
4. Transformación que se encarga de guardar los datos del XML de la petición en variables de DataPower para su posterior utilización. Esta transformación se podía haber incluido en la anterior, pero se ha dividido por razones de simplificación de código y para realizar un mayor control en las operaciones de los XSLT. DataPower permite comprobar el resultado de la ejecución del código para realizar pruebas e identificar errores en el código mediante la herramienta *Probe*, por ello es más recomendable realizar pequeñas modificaciones o transformaciones que incluirlo todo en un único fichero XSLT, más complejo y difícil de depurar. Al ser una transformación

para almacenar en variables determinados campos de la petición, el contexto de entrada es INPUT y el de salida es NULL.

5. Se realiza un XSLT para enviar los datos hacia el servicio web intermedio; en concreto, la finalidad de esta petición interna es la de obtener un dato de la respuesta, el código del carro de la compra o *cartId*, para realizar otra llamada al siguiente servicio web. Esta transformación tiene como contexto de entrada INPUT, y como contexto de salida PIPE. Esto es, al tener una salida datos, se debe indicar en qué contexto se almacenan para su posterior utilización. El contexto PIPE es un contexto especial que envía la información a la acción inmediatamente posterior de la regla. De este modo, al no crear un contexto nuevo con toda la información, se optimiza el rendimiento de las aplicaciones que utilizan esta regla.
   En el código XSLT de la transformación se monta la petición SOAP que se va a ejecutar en la llamada teniendo en cuenta que se debe respetar siempre el acuerdo de interfaz. Como parámetro de entrada se utiliza el valor de la variable de DataPower, guardado en la acción anterior, de nombre *customerId*. El detalle de este código se muestra a continuación:

```
<xsl:template match="/">
          <!-- con el valor de las variables obtenidas antes se prepara el xml a enviar al
primer servicio -->
          <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://predic8.com/WSDL/shop/1/">
                    <soapenv:Header/>
                    <soapenv:Body>
                      <ns:create>
                        <customerId>
                    <xsl:value-of select="dp:variable('var://context/var1/customerId')"/>
                        </customerId>
                      </ns:create>
                    </soapenv:Body>
          </soapenv:Envelope>
</xsl:template>
```

6. Acción de tipo *Result m*ediante la cual se realiza una petición a un sistema externo definiendo un protocolo determinado[90]. Para los anteriores servicios web representados, sólo indicaba el envío al destino final, no un destino intermedio, por ello no se modificaba su configuración. Para este caso, al ser una llamada HTTP intermedia, es necesario especificarlo. Se configura el campo *destination* de la acción indicando el destino con el protocolo *http:///,* por ser dentro de la red de confianza. Cabe también la posibilidad de realizar la llamada o ejecución del servicio externo mediante una transformación XSLT (un *extension element*[91]), pero se opta realizarlo con una acción específica por ser más limpio en cuanto a visualización y depuración de código. En esta acción el contexto de entrada es PIPE, para recibir datos de la acción anterior. El de salida es un contexto no predefinido, con la finalidad de almacenar los datos de la respuesta para luego poder acceder en acciones posteriores. Se nombra el contexto como *dpvar_1*.

[90] Acción ***Result*****:** es un tipo de acción definida en DataPower y utilizado en la definición de las reglas de las políticas de tratamiento de transacciones. Permite definir la localización de un recurso, definido como destino, especificando su URL o referenciando la variable que lo contenga. El modo de comunicación puede ser síncrono, si se espera respuesta con un timeout definido, o asíncrono, si se continúa ejecutando la regla independientemente de si se obtiene o no respuesta. Con ello, permite definir un tiempo de timeout, si existen reintentos, y el tiempo entre reintentos. Para conexiones de tipo HTTP permite definir el método a utilizar (POST, GET, PUT, DELETE, HEAD).

[91] Se utilizaría la extensión de DataPower ***dp:url-open*****.** Proporciona la posibilidad de enviar o recibir datos de cualquier URL utilizando un protocolo determinado.

7. Mediante esta transformación posterior a la llamada HTTP, se habilita el guardado de los datos de la respuesta XML en la base de datos. Se va a guardar en la tabla DATOS, realizando un XSLT que recorra el árbol del XML de respuesta e insertando los campos en la misma. Con esta transformación, para el caso de la solución presentada, se está almacenando el valor del campo *cartId.* Por ello el contexto de entrada es *dpvar_1* y el de salida NULL.
8. Otra transformación sucesiva que almacena el valor de los campos devueltos en variables de DataPower, de manera similar que el paso cuarto de esta regla. El contexto de entrada es *dpvar_1* y el de salida NULL.
9. La siguiente transformación prepara la llamada al último servicio web de nuestra red interna. Para este caso se realiza la llamada al de añadir al carro los artículos. Se utiliza el *cartId* obtenido anteriormente junto con los campos proporcionados en la llamada inicial. El concepto de implementación de esta transformación es similar al del paso 5 pero adaptándolo al WSDL mostrado en la [tabla 3.10]. El contexto de entrada es *dpvar_1* y el de salida es PIPE.
10. Con esta acción se envía el resultado de la transformación anterior al destino, por lo que el contexto de entrada es PIPE y el de salida es OUTPUT. Este último contexto define el de salida del servicio web que está definido en el *EndPoint* del *Web Service Proxy.*

En lo que respecta a la definición de la **regla de salida**, o en sentido respuesta, la situación es más sencilla. La segunda regla mostrada en la [figura 3.28] representa las acciones para la misma. Para esta implementación sólo es necesario recoger la respuesta y preparar el XML de salida de forma que se cumpla el acuerdo de interfaz definido en el WSDL del *Web Service Proxy.* Además, se incluye firmado de la respuesta, enviando el XML al origen firmado mediante el certificado de la Empresa A. Todo es muy similar al resto de reglas definidas para los servicios web desarrollados como ejemplo.

Para este WSP se ha definido también la **regla de error** con el fin de poder mostrar el funcionamiento de una de ellas. Esta regla es muy similar, en concepto, a la regla de salida, aunque sólo se ejecuta cuando se captura algún error de tipo técnico. Cuando ocurre, la ejecución de la regla actual se para y se comprueba si existe alguna regla de error que procese la transacción. En este caso se ha configurado para que procese todos los tipos de error, definiendo un filtrado para todas las órdenes en la acción *Match* inicial. Los errores de tipo técnico corresponden a indisponibilidad de acceder a destino, *timeout*, error al "*parsear*" o transformar las respuestas, etc. Un error de tipo funcional, como por ejemplo no existe cliente, no se va a capturar, devolviendo la respuesta al origen. Posteriormente, en las acciones de la regla, se almacenan los datos de error en base de datos y se devuelve la petición firmada. En este caso no es necesario definir una acción de tipo *Result* como en el resto de reglas, ya que únicamente se devuelve al origen por ser una tarea de error.

f) Creación y validación de la firma de las transacciones

Como se ha podido comprobar en el apartado anterior, la firma de las peticiones se realiza dentro de las reglas cada vez que se recibe o se devuelve una comunicación con el origen de la transacción. Para las peticiones realizadas de manera interna por DataPower no es necesario firmar ya que se están realizando dentro de la red de confianza. En caso de tener peticiones hacia redes externas, sí es muy recomendable definir comunicación SSL y firma.

g) Trazas de la transacción en Base de Datos

Las trazas en base de datos se han gestionado de manera similar a los casos anteriores, realizando un desarrollo XSLT a medida para la lectura en bucle de los datos de respuesta. El problema reside en que es necesario realizar a medida este tipo de tratamientos, modificando el código para tratar el formato de la repuesta correctamente. Sin embargo, el concepto sí es similar para todos estos casos: realizar el guardado de manera recursiva. Esto favorece además a que cambios menores en el WSDL, tales como añadir parámetros nuevos, o cambiar el tipo de datos, no afecten a las trazas de DataPower.

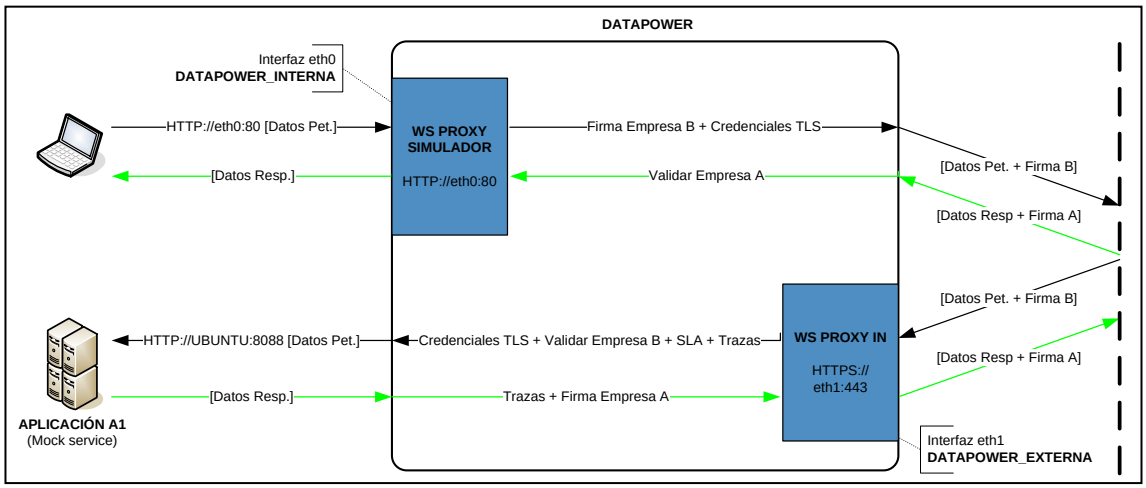
Todos los datos devueltos por los servicios web van a insertarse en la tabla DATOS, así como el inicio y el fin en la tabla TRAZAS. En este caso se realizan dos inserciones adicionales en la tabla DATOS debido a las llamadas realizadas por DataPower a los servicios web externos.

4 Pruebas y resultados

4.1 Comunicación extremo a extremo: sentido entrada

La realización de las pruebas implica que exista una arquitectura completa punto a punto. Para ello, puesto que el diseño se ha tenido en cuenta siempre desde la parte de la Empresa A, se necesita implementar un proceso adicional que simule las peticiones y respuestas que genera la Empresa B. Dicho proceso se ha llamado "simulador" y se ha implementado utilizando DataPower, más concretamente, un objeto de tipo *Web Service Proxy.* Se ha optado por realizar este desarrollo adicional mediate DataPower para tener unificada la tecnología utilizada para las pruebas. Con ello se minimizan los posibles puntos de fallo, consiguiendo una arquitectura para pruebas más fiable, robusta y compacta. Esto es muy importante ya que así no se depende de sistemas externos para realizar las pruebas. Como limitación a esta elección, la relativa complejidad al realizar llamadas entre distintos WS dentro del mismo aplicativo y la necesidad, en ocasiones, de realizar algún tipo de tratamiento XML mediante XSLT.

Para el caso de una comunicación en sentido entrada a la red, el simulador proporciona la función de inyectar peticiones desde nuestra red local como si fueran generadas desde el exterior por la Empresa B. Lo más sencillo es realizar un enlace o puente entre la dirección de "DATAPOWER_INTERNA", en la que se inyecta la petición de inicio manualmente, y la de "DATAPOWER_EXTERNA", en la que se encuentra levantado el servicio web de entrada.



[Figura 3.29] Diagrama de arquitectura para la realización de pruebas de un servicio web en sentido entrada

Como puede observarse en la [figura 3.29], las peticiones entrantes al servicio web de la Empresa A se simulan realizando una llamada SOAP HTTP a la IP interna de DataPower (*eth0)* con los datos. En esa IP y puerto está levantado el simulador de la Empresa B en DataPower, que en este caso es el inicio del flujo de negocio. Dentro de la política de reglas del simulador, se realiza la firma como Empresa B y se utilizan las credenciales de ésta para realizar la conexión TLS al servicio web de la Empresa A. Este es el servicio web que se ha ido explicando en los pasos anteriores y que se inicia con la llegada de una petición a la dirección HTTPs externa, "DATAPOWER_EXTERNA". Con los datos y la firma de la Empresa B que han llegado, se ejecutan las reglas del servicio web de entrada: se validará la firma, extraerán los datos y se inyectarán en la aplicación de la Empresa A de nuestra red local para obtener una respuesta. La aplicación se crea como un proceso de tipo *Mock* generado con SoapUI. Una vez se tenga la respuesta, se envía firmada por la Empresa A al origen. Todo ello se realiza implementado trazas en base de datos. Por su parte, el simulador de la

Empresa B debe validar la firma de la Empresa A y presentar los datos recuperados al usuario que le realizó la llamada. Por lo tanto, simulando esta comunicación extremo a extremo se puede comprobar que se cumplan todos los requisitos.

El resumen de la batería de pruebas se muestra en la siguiente tabla.

REQUISITO	PRUEBA	RESULTADO
Credenciales de la Empresa A	Verificación de la identidad proporcionada por el aplicativo cuando se accede a la dirección HTTPs y URI.	**Correcto**: certificado de la Empresa A
Credenciales de la Empresa B	Inyectar peticiones (SoapUI) al simulador de DataPower que se presenta con las credenciales de la Empresa B para realizar el *handshake*.	**Correcto**: comunicación establecida
Credenciales de la Empresa B	Inyectar peticiones (SoapUI) al simulador de DataPower que se presenta con las credenciales no verificadas en el servicio web para realizar el *handshake*.	**Correcto**: no se puede establecer comunicación
Cumplimiento de SLA: umbral de 200 peticiones en intervalo de un día	Envío de más de 200 peticiones en una franja horaria de 24 horas.	**Correcto parcialmente**: las peticiones que superen el umbral se rechazan. **(* ver detalle)**
Cumplimiento de SLA: umbral de 20 peticiones en intervalo móvil de una hora	Envío de más de 20 peticiones en un intervalo de menos de una hora. Luego enviar una petición cada cierto tiempo hasta superar el intervalo de una hora.	
Trazas de aplicativo en servidor (syslog)	Inyectar peticiones (SoapUI) al simulador. Herramienta *Probe* en DataPower activada para captura debug. Comprobación en el fichero de texto del servidor UBUNTU de las transacciones capturadas por el *Probe.*	**Correcto**: se observa en el fichero de texto la referencia a las transacciones capturadas.
Trazas de aplicativo en base de datos: tabla TRAZAS	Envío de transacciones de prueba al simulador.	**Correcto**: verificación de los datos de inicio y fin en la tabla TRAZAS.
Trazas de aplicativo en base de datos: tabla DATOS	Envío de transacciones de prueba al simulador con distintos datos a devolver.	**Correcto**: verificación de los datos de la respuesta en la tabla DATOS.
Transacción correcta	Envío de transacción con formato de WSDL correcto. Aplicación de la Empresa A devuelve valor correcto de *id.*	**Correcto**: transacción con respuesta correcta devuelta y trazabilidad completa
Transacción con error funcional	Envío de transacción con formato de WSDL incorrecto.	**Correcto:** al no cumplir el WSDL de entrada se rechaza la petición sin inserción en BD, únicamente con un log informativo de sistema en el fichero del servidor UBUNTU.
Transacción con error funcional	Envío de transacción con formato de WSDL correcto. Aplicación de la Empresa A devuelve un valor incorrecto en campo *id*	**Correcto parcialmente**: **(** ver detalle)**
Transacción con error técnico	Envío de transacción con formato de WSDL correcto. Aplicación de la Empresa A no está accesible.	
Velocidad en el tratamiento de las transacciones	Revisión de tiempo de tratamiento de las transacciones de la batería de pruebas ejecutadas.	**Correcto**: la latencia de las peticiones individuales no supera los 70 ms.
Velocidad en el tratamiento de las transacciones	Revisión de la latencia con 20 peticiones concurrentes, el máximo para las peticiones a la hora.	**Correcto**: la latencia total asciende a 260 ms, de los cuales 120 ms corresponden a la latencia del sistema final (UBUNTU). **(*** ver detalle)**

[Tabla 3.11]: Resumen de pruebas realizadas para la comunicación punto a punto en sentido entrada

(*) Detalle de las pruebas de SLA

En lo que respecta a la medición de este requisito, nos vamos a centrar, de los dos umbrales configurados en DataPower, en el relativo al umbral móvil de una hora. La elección de uno u otro para definir el funcionamiento no es determinante, ya que se han definido bajo el mismo objeto de medición SLM. El escoger el umbral relativo al intervalo móvil en una hora es una elección meramente práctica, por tener un tiempo de definición menor. Posee además una ventaja respecto al umbral de peticiones diaria: el intervalo definido en este caso es móvil, no es un intervalo fijo predefinido. Por lo tanto, estudiando el comportamiento

del umbral móvil en una hora se define el comportamiento general en cuanto a medición de los SLA.

El modo en el que se ha medido el requisito del umbral en el tramo de una hora es enviando una sucesión de peticiones al servicio web con el siguiente patrón: dos peticiones aisladas y luego una sucesión de dos peticiones por minuto. Una vez que se llega, en menos de una hora, al límite de 20 peticiones, el resto que se van recibiendo son rechazadas. La respuesta SOAP que se obtiene por el rechazo a causa de superar el límite es:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Body>
      <env:Fault>
         <faultcode>env:Server</faultcode>
         <faultstring>No signature in message! (from server)</faultstring>
      </env:Fault>
   </env:Body>
</env:Envelope>
```

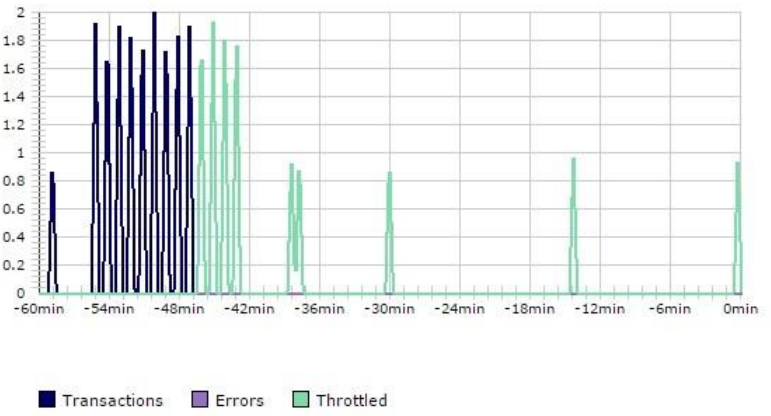
Como se puede observar, el error devuelto muestra que no existe firma en el mensaje recibido. Este error se genera porque el servicio web de la Empresa A está limitado a 20 peticiones en el intervalo de 1 hora. Si se supera, se para la ejecución de las acciones posteriores de la regla y se devuelve al origen un error de tipo *"Rejected by SLM Monitor".* Como la firma de la petición por parte de la Empresa A no se realiza por ser posterior a la medición de SLM, en el simulador no se puede verificar la firma de la respuesta recibida y, por consiguiente, devuelve error, que es el mostrado en el código anterior. Por lo tanto, es un error concatenado derivado de no tener una regla de gestión de errores en el servicio web de la Empresa A. El error que hemos obtenido es el que ha ocurrido dentro del simulador, no el ocurrido por el rechazo de la acción SLM. Este tipo de problemas se soluciona creando una regla de errores tal y como se ha creado para el servicio web que se comporta como un ESB. En la regla de errores se incluiría la firma de la petición, con lo que el simulador obtendría el error en bruto del servicio y sería capaz de devolverlo al origen.

Para este tipo de rechazos no se inserta ninguna petición en base de datos, los mismos se deben ver en las trazas de sistema, como por ejemplo:

```
        Jul  8 20:53:24 DP_SERGIO [0x80c00010][xslt][error] wsgw(WS_Proxy_IN):
trans(56368)[request][192.168.2.118] gtid(49106): Processing of 'store:///dp/slmpolicy.xsl'
stopped: Rejected by SLM Monitor
```

A continuación, en la [figura 3.30], se muestra una gráfica obtenida con DataPower en la que se muestran las peticiones enviadas en un margen de tiempo. El eje Y representa el número de peticiones recibidas, mientras que el eje X es la escala de tiempo, que se establece en una hora. Se representan tanto las que se procesan correctamente, en la leyenda como *Transactions,* como las que se rechazan por no cumplir el SLA, *Throttled.* No se simulan peticiones de error por deber simular la medición del SLA, ya que si el error ocurre antes de la acción de medición de umbrales, no se ejecutaría y falsearía los datos. Por lo tanto se parte de la situación óptima para la medición.

Dentro de la escala de tiempo mostrada en la figura, las peticiones nuevas que superen el umbral de las 20 permitidas, son rechazadas. Esto va a mantenerse hasta que la ventana de una hora contiene menos de 20 peticiones, por lo que se aceptarán las peticiones que resten hasta el umbral de 20 por haberse ido más allá de la ventana móvil de una hora.



[Figura 3.30]: Peticiones recibidas por el servicio web en una hora

Por lo tanto, el funcionamiento de la medición de los SLA es correcto, sin embargo el no definir el error correcto a la vuelta enmascarándolo en uno de otro tipo, implica que se catalogue la prueba como **correcta parcialmente**.

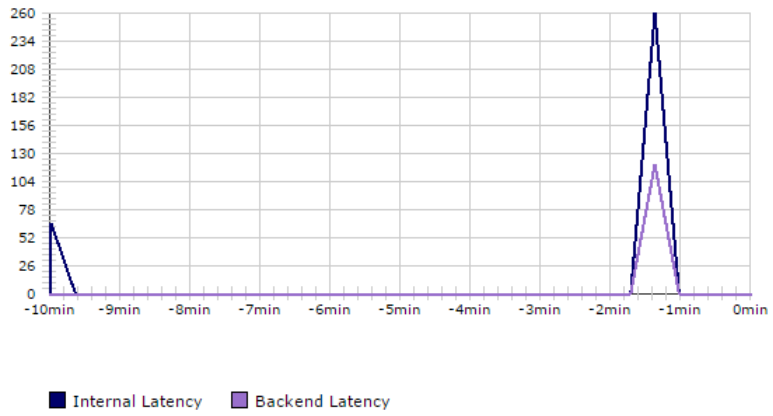
(**) Detalle de las pruebas de error técnico y funcional

Para el caso de la simulación de errores, la implementación reacciona como se espera. Sin embargo no es la forma correcta de proceder para cualquier desarrollo genérico en DataPower, ya que siempre se debe realizar una regla de tratamiento de las transacciones en caso de error. Al no realizarla, como en este caso, los errores son devueltos hacia el origen sin realizar ningún tipo de tratamiento, lo cual puede provocar errores concatenados como los ocurridos en las pruebas anteriores. Para este caso no es una excepción, devolviendo esta prueba el mismo error que la anterior, al no ser firmada tampoco la petición de respuesta hacia la Empresa B: "*No signature in message! (from server)*". Para ambos casos, errores técnico y funcional, se devuelve ese tipo de error y, también para ambos casos, sólo se inserta el inicio de la petición en las trazas de base de datos, no el fin. Esto provoca que las trazas en base de datos queden incompletas, no mostrando la fecha real en la que se devolvió el error.

El resultado de la prueba se estima como **correcto parcialmente**, por responder como era de esperar pero por no ser la solución óptima.

(***) Detalle de las pruebas de carga concurrente

Para poder poner a prueba la velocidad de procesamiento de DataPower se envían de manera concurrente varias peticiones, por lo que DataPower debe gestionarlas todas en el mismo intervalo de tiempo. En este caso se puede afirmar que los resultados no son concluyentes. El rendimiento de DataPower, al ejecutarse como una máquina virtual, siempre va a ser menor que el de la máquina física. Además, el tener la utilidad *Probe* de DataPower activada para la captura de trazas, como ya se comentó antes, penaliza el rendimiento. Por lo tanto, se puede concluir que se está partiendo de la peor situación posible para su medición: son máquinas virtuales ejecutándose sobre una sóla máquina física con características de nivel usuario, y con las herramientas de *debug* activadas en DataPower. Aún en esta situación, la respuesta de las mismas es bastante satisfactoria, como se puede ver en la siguiente gráfica.



[Figura 3.31] Peticiones recibidas por el servicio web en una hora

En el eje X se mide un intervalo de tiempo, en este caso de 10 minutos. En el eje Y se muestra la latencia de las transacciones en milisegundos. Como se puede comprobar, la latencia de las 20 transacciones ejecutadas simultáneamente no supera los 260 ms en total. Dentro de ese tiempo se incluye la latencia generada por el procesamiento del backend, que en este caso es de unos 120 ms, aproximadamente el 45% del total. Por lo tanto, aún en estas situaciones de carga, DataPower es capaz de procesar correctamente las peticiones, insertando en la base de datos y enviando al destino sin retrasos.

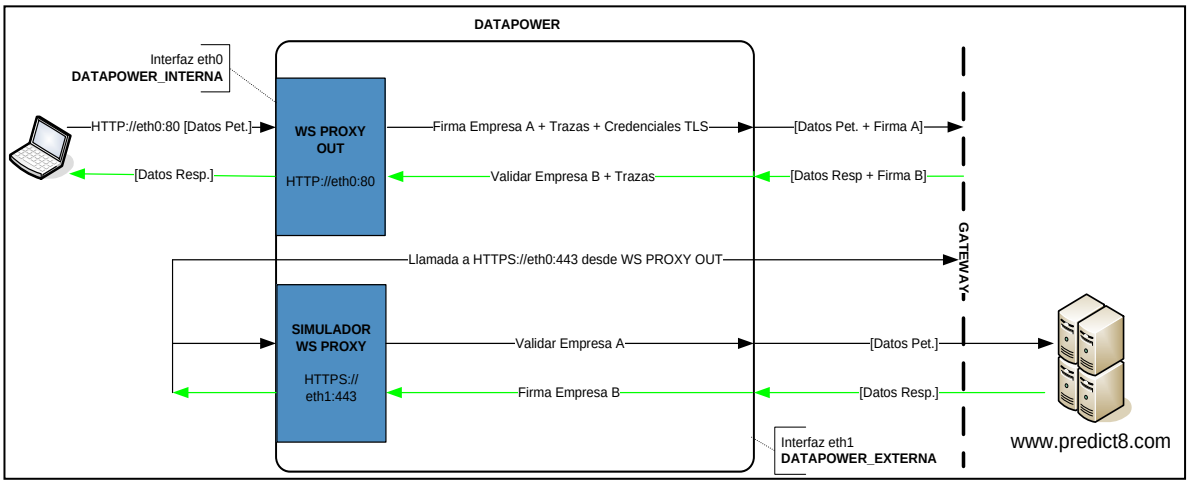
4.2 Comunicación extremo a extremo: sentido salida

Del mismo modo que el caso anterior, se realiza un simulador utilizando las herramientas de DataPower para implementar el comportamiento de la Empresa B. Al ser el sentido de la comunición hacia una red exterior, es necesario modificar el destino del servicio de salida (el servicio web que se ha estado implementando) para que lo haga a una dirección de la red local. En esta dirección estará implementado el simulador mediante un *Web Service Proxy,* como en las pruebas anteriores.

Tanto el servicio implementado para una comunicación en sentido salida como el simulador *Web Service Proxy* de la Empresa B del apartado anterior, utilizan una conexión HTTP como punto de entrada en DataPower. Ambos están levantados sobre la misma IP y puerto al reutilizar el mismo objeto de DataPower *Front Side Handler* (*"http://"DATAPOWER_INTERNA":80"*). Lo que no comparten ambos servicios web es la URI de ejecución, por lo que ese va a ser el factor diferenciador para realizar la petición hacia uno u otro.

El simulador debe realizar varias acciones relativas a la Empresa B. Debe validar que se envíe la petición desde la Empresa A y debe firmar las respuestas con el certificado de la Empresa B. La respuesta en este caso no se va a generar mediante un *Mock Service*, sino que se va a utilizar una llamada al exterior dinámica. Se utiliza el servicio web público de la referencia ([48] WSDL del servicio utilizado para una comunicación extremo a extremo). De este modo el formato de campos de respuesta depende del Servicio Web que está disponible en Internet, acercándonos a un caso más real por estar accediendo a una red externa.

El diagrama mostrado a continuación muestra las acciones básicas de cada servicio involucrado en la simulación.



[Figura 3.32] Diagrama de arquitectura para la realización de pruebas de un servicio web en sentido salida

A continuación se muestra la tabla resumen de las pruebas ejecutadas. Para esta ocasión no es necesario probar el nivel de servicio o SLA ya que en la implementación no existe.

REQUISITO	PRUEBA	RESULTADO
Credenciales de la Empresa A	Inyectar peticiones con SoapUI. Verificación de las trazas de salida del servicio web mediante la herramienta *Probe*.	**Correcto**: Visualización de la firma correcta para la Empresa A. *Handshake* con el simulador de DataPower correcto.
Credenciales de la Empresa B	Inyectar peticiones (SoapUI) al servicio web de DataPower.	**Correcto**: comunicación establecida. Se recibe respuesta del servicio en red externa.
Trazas de aplicativo en servidor (*syslog*)	Inyectar peticiones (SoapUI). Herramienta *Probe* en DataPower activada para captura debug. Comprobación en el fichero de texto del servidor UBUNTU de las transacciones capturadas por el *Probe.*	**Correcto**: se observa en el fichero de texto la referencia a las transacciones capturadas.
Trazas de aplicativo en base de datos: tabla TRAZAS	Envío de transacciones de prueba al simulador.	**Correcto**: verificación de los datos de inicio y fin en la tabla TRAZAS.
Trazas de aplicativo en base de datos: tabla DATOS	Envío de transacciones de prueba al simulador.	**Correcto**: verificación de los datos de la respuesta en la tabla DATOS.
Transacción correcta	Envío de transacción con formato de WSDL correcto.	**Correcto**: transacción con respuesta correcta devuelta y trazabilidad completa
Transacción con error funcional	Envío de transacción con formato de WSDL de entrada al *web service* de DataPower incorrecto.	**Correcto:** al no cumplir el WSDL de entrada se rechaza la petición sin inserción en BD, únicamente con un log informativo de sistema en el fichero del servidor UBUNTU.
Transacción con error funcional	Envío de transacción con formato de WSDL correcto.	**Correcto parcialmente:** al igual que para las pruebas del servicio anterior, el resultado es correcto pero no óptimo. La respuesta no indica el error real devuelto por el servicio web.
Transacción con error técnico	Envío de transacción con formato de WSDL correcto. Aplicación externa no está accesible, se apunta a otro destino desde el simulador.	
Velocidad en el tratamiento de las transacciones	Revisión de tiempo de tratamiento de las transacciones de la batería de pruebas ejecutadas.	**Correcto:** el tiempo medio de tratamiento no supera los 70 ms.
Velocidad en el tratamiento de las transacciones	Revisión de la latencia con 20 peticiones concurrentes.	**Correcto:** el tiempo medio de tratamiento no supera las 200 ms.

[Tabla 3.12]: Resumen de pruebas realizadas para la comunicación punto a punto en sentido salida

4.3 Transferencia de ficheros

Para las pruebas de transferencia de ficheros se utiliza un servidor de ficheros proporcionado por un proveedor de acceso a Internet, de uso privado y personal, en una red externa, y un servidor en la máquina UBUNTU en la red interna. DataPower será el encargado de comunicar ambos servidores de FTP.

De cara a comprobar el funcionamiento es necesario revisar el estado de los logs de sistema y de las trazas en base de datos de las ejecuciones. El intervalo de recuperación de ficheros del servidor externo se configuró a un minuto para poder realizar las pruebas de obtención del mismo más rápido.

Nuestro proceso sólo va a comenzar cuando el *Poller* o temporizador que inicia la acción de comprobar el fichero lo encuentre en el origen. Puede observarse que lo ha recuperado cuando se escribe en base de datos la correspondiente fila en la tabla TRAZAS. Para observar las trazas en base de datos se utiliza SQLPLUS, aunque se puede utilizar también alguno más completo como *SQL Developer*[92]. A continuación se muestra la query resultante de la búqueda realizada con SQLPLUS:

```
        SQL> select * from (select * from trazas where servicio='SFTP' order by fecha_inicio
desc) where rownum < 2;
             467059 SFTP
        DP_SERGIO
        14/05/15 19:43:38,262352
        14/05/15 19:43:38,262352
        OK
        test1.zip?Rename=test1.zip.DP_SERGIO.0000000.desarrollo.FTP_Poller_Orange.143157
        4361
```

Con la query anterior se obtiene el último registro de ejecución del proceso de transferencia de ficheros. Para buscar la traza de sistema generada se utiliza el campo TRANSACCION de la respuesta anterior, como se muestra más adelante:

```
        sgomezpa@ubuntu:/var/log$ grep 467059 datapower.log
        May 14 19:43:38 DP_SERGIO [0x00340027][multistep][warn] mpgw(SFTP_Poller_Empresa_B_A):
trans(467059)[request]: Multistep Probe enabled
        May 14 19:43:38 DP_SERGIO [0x00340027][multistep][warn] mpgw(SFTP_Poller_Empresa_B_A):
trans(467059)[response]: Multistep Probe enabled
```

El resultado de la ejecución en este caso ha sido satisfactorio, ya que el fichero en el servidor origen está renombrado a *"test1.zip.processed.ok"*. Además puede observarse en el directorio destino:

```
        sgomezpa@ubuntu:/home/datapowersftp/work$ ls -rlt
        total 540
        -rw-r--r-- 1 datapowersftp ftpaccess 549725 may 14 19:43 test1.zip
```

En el caso en que no exista el fichero, se obtendría un error en las trazas de sistema. No se escribiría en la base de datos ya que la política de actuación del *Multi-Protocol Gateway*, en la que se encuentra la escritura de las trazas, sólo se ejecutaría en caso de

[92] **SQLPLUS y *SQL Developer*** son dos herramientas de Oracle para el manejo de sus bases de datos. En el caso de SQLPLUS, se dispone de una interfaz de línea de comandos para realizar consultas utilizando SQL y PL/SQL. *SQL Developer* es más completo ya que es un entorno de desarrollo que permite además gestionar las bases de datos. Ambos productos están disponibles de manera gratuita desde la página oficial de Oracle [55]

entrada de fichero. Se puede observar el siguiente error dentro del fichero de trazas de DataPower de la máquina UBUNTU:

```
        May 14 08:21:43 DP_SERGIO [0x80e000d2][file-poller][notice] source-ftp-
poller(FTP_Poller_Orange): trans(36079): No input found for URL 'ftp://ftp.orange.es/'
```

Un problema que se observó fue el configurar de manera incorrecta el *BackEnd* sin especificar el nombre del fichero en la ruta completa. Al no indicarlo, DataPower no puede localizar el destino del fichero obtenido y se alertaría con errores de tipo *backside routing*:

```
        Apr 24 16:31:25 DP_SERGIO [0x80e00126][mpgw][error] mpgw(SFTP_Poller_Empresa_B_A):
trans(78290): Valid backside connection could not be established: SFTP: Failed to perform
requested operation, url: sftp://192.168.2.113/work
```

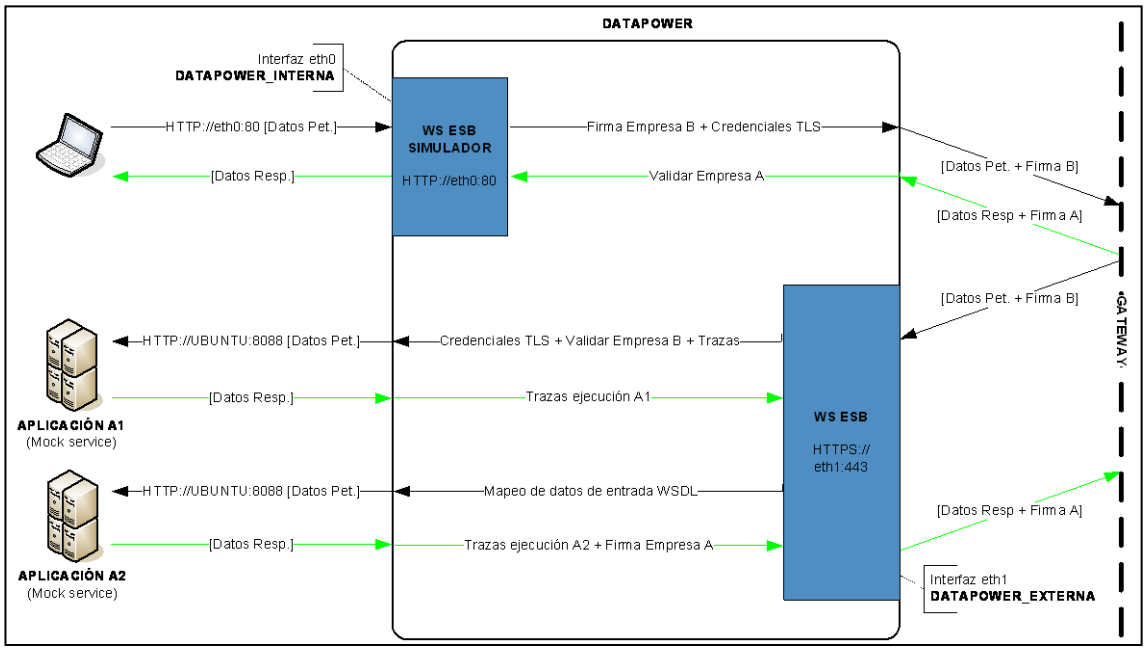
En el caso de que ocurra un error de este tipo, no se ha definido una política para el tratamiento de los mismos, por lo que también se advertirá en los logs de datapower:

```
        Apr 24 16:31:25 DP_SERGIO [0x80c0007b][mpgw][warn] stylepolicy(SFTP_Server_Policy):
trans(78290): No error rule is matched.
```

Tras la finalización de las pruebas, comprobando que el fichero se obtiene desde un servidor externo y se envía a uno interno mediante DataPower, se puede certificar que **la implementación se comporta correctamente.**

4.4 ESB – Enterprise Service Bus

La [figura 3.33] describe el escenario de pruebas de esta aplicación.



[Figura 3.33] Diagrama de arquitectura para la realización de pruebas de un servicio web de tipo ESB

Para las pruebas de esta implementación se realiza de nuevo un simulador en DataPower con el objetivo de implementar la funcionalidad que realiza la Empresa B para el envío y recepción de las transacciones hasta nuestra Empresa A. Eso es, se crea un servicio web levantado en la IP "DATAPOWER_INTERNA" y que realice una llamada a la IP "DATAPOWER_EXTERNA", en la que se reciben las peticiones de la Empresa B.

La batería de pruebas a realizar se puede comprobar en la siguiente tabla. En este caso sí que existe una regla para el tratamiento de errores, por lo que a priori es de suponer que se comporte de una manera más completa que los anteriores. No se ha implementado en este caso la medición de SLA proporcionada con DataPower, por lo que esas pruebas no van a resultar necesarias.

REQUISITO	PRUEBA	RESULTADO
Credenciales de la Empresa A	Verificación de la identidad proporcionada por el aplicativo cuando se accede a la dirección HTTPs y URI.	**Correcto**: certificado de la Empresa A
Credenciales de la Empresa B	Inyectar peticiones (SoapUI) al simulador de DataPower que se presenta con las credenciales de la Empresa B para realizar el *handshake*.	**Correcto**: comunicación establecida. Se recibe respuesta, por lo que el simulador y el servicio web han realizado el *handshake* correctamente.
Credenciales de la Empresa B	Inyectar peticiones (SoapUI) al simulador de DataPower que se presenta con las credenciales no verificadas por el servicio web.	**Correcto**: no se puede establecer comunicación, el certificado utilizado por el simulador no está permitido.
Trazas de aplicativo en servidor (*syslog*)	Inyectar peticiones (SoapUI) al simulador. Herramienta *Probe* en DataPower activada para captura debug. Comprobación en el fichero de texto del servidor UBUNTU de las transacciones capturadas por el *Probe.*	**Correcto**: se observa en el fichero de texto la referencia a las transacciones capturadas.
Trazas de aplicativo en base de datos: tabla TRAZAS	Envío de transacciones de prueba al simulador.	**Correcto**: verificación de los datos de inicio y fin en la tabla TRAZAS.
Trazas de aplicativo en base de datos: tabla DATOS	Envío de transacciones de prueba al simulador con distintos datos a devolver.	**Correcto**: verificación de los datos de la respuesta en la tabla DATOS.
Transacción correcta	Envío de transacción con formato de WSDL correcto. Aplicación de la Empresa A devuelve valor correcto de *id.*	**Correcto**: transacción con respuesta correcta devuelta y trazabilidad completa
Transacción con error funcional	Envío de transacción con formato de WSDL incorrecto.	**Correcto:** al no cumplir el WSDL de entrada se rechaza la petición sin inserción en BD, únicamente con un log informativo de sistema en el fichero del servidor UBUNTU.
Transacción con error funcional	Envío de transacción con formato de WSDL correcto. Aplicación A1 para creación de carro devuelve formato incorrecto.	**Incorrecto:** se provoca que A1 devuelva una respuesta SOAP no contemplada en el WSDL, aun así la llamada se tramita normalmente, insertando una traza de OK en base de datos y respondiendo a origen con una traza de OK. **(* Ver Detalle)**
Transacción con error funcional	Envío de transacción con formato de WSDL correcto. Aplicación A2 para añadir al carro devuelve formato incorrecto.	**Correcto:** Se recibe un error funcional indicando fallo en el acuerdo del WSDL. Inserción correcta en base de datos.
Transacción con error técnico	Envío de transacción con formato de WSDL correcto. Aplicación A1 no está accesible.	**Correcto:** se recibe un error al intentar procesar la respuesta de la aplicación A1, ejecutando la correspondiente regla de error e insertando los datos en los logs de BD.
Transacción con error técnico	Envío de transacción con formato de WSDL correcto. Aplicación A2 no está accesible.	**Correcto:** se recibe un error al intentar procesar la respuesta de la aplicación A1, ejecutando la correspondiente regla de error e insertando los datos en los logs de BD.
Transacción con error técnico	Envío de transacción con formato de WSDL correcto. Se para la Base de Datos Oracle para logs, por lo que no está accesible.	**Correcto:** las órdenes se ejecutan con total normalidad, no se ven afectadas por posibles caídas o fallos de acceso a la base de datos de logs.
Transacción con error técnico	Envío de transacción con formato de WSDL correcto. Fichero de logs de sistema no está accesible/ corrupto.	**Correcto** Las ejecuciones en DataPower se comportan con normalidad, no se ven afectadas.
Velocidad en el tratamiento de las transacciones	Revisión de tiempo de tratamiento de las transacciones de la batería de pruebas ejecutadas.	**Correcto**: la latencia de las peticiones individuales no supera los 105 ms medidos en base de datos, restando fecha de fin y de inicio.

[Tabla 3.13]: Resumen de pruebas realizadas para un servicio web funcionando como un ESB

(*) Detalle de las pruebas de error funcional para aplicación A1

El problema ocurre cuando se envía al servicio web de la aplicación A1 el alta del carro. El acuerdo de interfaz de campos es el mostrado en la [tabla 3.9]. Al añadir en la respuesta del servicio un campo adicional, incumpliendo así el WSDL que hemos definido en la aplicación A1, DataPower la recibe y la trata como si estuviera correcta. Esto es, no realiza ninguna validación de campos de respuesta, solamente envía en el formato correcto y espera una respuesta independientemente de su contenido.

Para corroborarlo se ha configurado en el servicio de la aplicación A1, el proceso *MockService* de SoapUI, la siguiente respuesta con un campo adicional llamado "desc".

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:createResponse xmlns:ns2="http://predic8.com/WSDL/shop/1/">
         <cartId>Cart-00048</cartId>
         <desc>Campo incorrecto, añadido para fallo funcional</desc>
      </ns2:createResponse>
   </S:Body>
</S:Envelope>
```

No se recibe ningún error, es más, el campo adicional se inserta en la base de datos como si de uno correcto se tratara. El motivo de este comportamiento es el no haber realizado una validación de la respuesta intermedia una vez recibida. Se debe añadir otra acción de tipo *Validate* posterior a la respuesta del servicio web en la regla en DataPower:  . Dentro de la misma se le debe especificar un WSDL mediante el cual se valida la respuesta. El WSDL a utilizar es el relativo al servicio web que se invoca de la aplicación A1, el de creación de carro.

Una vez añadida la validación, se procede de nuevo a realizar de nuevo la prueba, resultando en este caso satisfactoria. El siguiente código muestra el xml devuelto, antes de firmar el XML como Empresa A, cuando existe un error de formato.

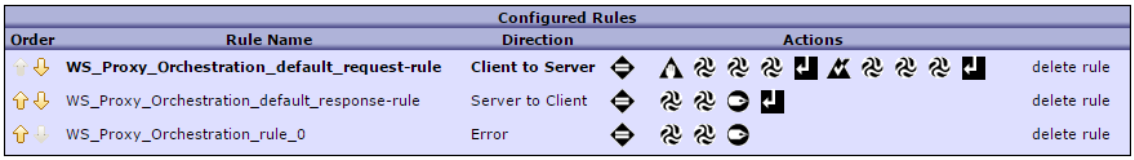
```
<soapenv:Envelope
xmlns:ns="http://datapower.com/WSDL/addToCartByCustomerId/1/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
>
<soapenv:Header />
<soapenv:Body>
<ns:addToCartResponse>
<cartId />
<error>0x00230001</error>
<description>http://UBUNTU:8089/mockShopServicePTBinding: cvc-complex-type 2.4: in element
{http://predic8.com/WSDL/shop/1/}createResponse of type
{http://predic8.com/WSDL/shop/1/}CreateResponseType, found <desc> (in default namespace), but
next item should be end-element</description>
</ns:addToCartResponse>
</soapenv:Body>
</soapenv:Envelope>
```

El código y la descripción, se insertarían en la base de datos, tanto en la tabla TRAZAS como en la tabla DATOS. Por lo que el comportamiento frente a errores funcionales de la aplicación A1 quedaría corregido.

Hay que destacar que este problema ocurría para las llamadas a la aplicación A1, la intermedia. Para la aplicación A2 no existía este problema a pesar de no especificar que validara los datos. El motivo es que la segunda llamada, la realizada a la aplicación A2, se

está utilizando en el fin de la regla de petición, por lo que la recepción de la respuesta va a implicar una validación del WSDL presentado al crear el *Web Service Proxy* antes de ejecutar la regla de vuelta. Por eso, para este caso, las pruebas que se han realizado son satisfactorias.

Por lo tanto, a modo de resumen, hay que incluir siempre una acción de validación en la regla de modo que para todas las llamadas intermedias a otros sistemas. Tras esta corrección, la regla del servicio web con un comportamiento como ESB quedaría tal como se muestra en la figura.



[Figura 3.34] Reglas para el servicio web ESB tras la corrección

Por lo tanto, una vez solucionado este problema descrito, se puede dar la totalidad de las pruebas como **correctas**.

A lo largo de ese apartado se han realizado pruebas con la finalidad de detectar posibles pautas anómalas en el comportamiento funcional de los desarrollos. Se ha intentado recrear situaciones comunes para poner en compromiso la funcionalidad de los desarrollos. Ante las mismas, sí se han encontrado discrepancias en cuanto al comportamiento, por lo que se han solucionado y se han explicado. Con ello se ha proporcionado un método de trabajo, con una batería de pruebas que podría ser común para los nuevos servicios web y con unas soluciones que pueden ayudar a identificar otros problemas en futuros desarrollos.

5 Conclusiones y líneas de trabajo futuras

5.1 Conclusiones

Tras la exposición completa del proyecto, cabe destacar las conclusiones fundamentales que se han obtenido del mismo.

En primer lugar, podemos afirmar que se han cumplido los objetivos prioritarios que se marcaron para realizar este proyecto: proporcionar una visión general de la utilización de DataPower como proxy en una comunicación y poder conocer los beneficios más importantes con ejemplos prácticos, lanzando además una batería de pruebas para cada escenario. Todos los casos son reutilizables y sirven como base para desarrollos más complejos.

Con la creación de este proyecto, se ha demostrado que DataPower es capaz de implementar un gran número de características relacionadas con la comunicación punto a punto. Esto proporciona muchas ventajas a las aplicaciones que lo utilizan y todo ello sin necesidad de recurrir a un código complicado. Sin embargo, conocer los detalles de todos y cada una de sus partes requiere del asentamiento previo de unos conocimientos base, los cuales se han ido obteniendo poco a poco según se avanzaba en el proyecto y surgían los primeros problemas. Con este proyecto se sintetiza toda la información disponible en la Web y en manuales del fabricante y se presenta de forma intuitiva; si bien es cierto que a veces es algo densa si no se poseen conocimientos previos, pero en ningún caso deja de ser útil. Los no iniciados en este tipo de tecnología pueden encontrar aquí su pequeño manual de inicio y aprendizaje.

A tenor del desarrollo del proyecto, se afianza la idea de una de las principales motivaciones del mismo, el proporcionar seguridad en las comunicaciones entre pares. Dado un problema, se escogió la solución integrada de DataPower para ponerle remedio. Con dicho aplicativo se ha conseguido proporcionar una capa de abstracción necesaria capaz de gestionar las transacciones de manera individual, incluyendo un tratamiento personalizado para cada una de ellas, y liberando así a los sistemas externos de la carga que genera que, además, implicaría un desarrollo adicional a veces no viable en las aplicaciones externas. Por lo que, gracias a seguir la línea impuesta por esta motivación, se han tratado temas como certificaciones, credenciales, firmas electrónicas, cifrado y verificación de identidades, proporcionando casos prácticos de uso que pueden aplicarse a los entornos productivos, y que completa la formación de conocimiento de este tipo de comunicaciones seguras.

Aunque durante el proyecto se han puesto de manifiesto los beneficios de DataPower, también se ha llegado a la conclusión de que esta solución no está recomendada para todos los casos o necesidades. En parte porque DataPower es una máquina muy potente, y sólo sistemas de comunicaciones que tengan una carga muy elevada de tráfico pueden rentabilizar el coste del mismo. Mediante estos los desarrollos realizados en DataPower, se ha observado que su utilización en las pequeñas empresas o con un pequeño número de aplicaciones o servicios no es recomendable. No se obtiene beneficio bruto de un sistema para el que se desaprovecha su potencial, es caro y sólo puede utilizarse para los fines concretos para los que se creó: la máquina de DataPower no se puede utilizar para instalar otro Sistema Operativo por ejemplo. Para una empresa que quiera adoptar esta solución, al coste de adquisición se debe sumar el coste de mantenimiento del sistema y de soporte del mismo por un equipo de personas especializado. Es decir, existe un compromiso entre la inversión a realizar para la mejora de un sistema y el beneficio que se va a obtener. Por lo tanto, el realizar una estimación para su implantación en sistemas reales es una necesidad básica a tener en cuenta.

Otra deducción interesante a la que se ha llegado realizando el desarrollo de los servicios web en DataPower, es el de añadir de forma fácil nuevas funcionalidades sin afectar a la que está implementada. Esto quiere decir que el sistema es escalable y reutilizable. Muchos objetos creados se han podido reutilizar para la creación de otros nuevos. Teniendo

el aplicativo correctamente configurado y debidamente actualizado en cuanto a firmware se refiere, permite disponer de una solución reutilizable para las nuevas aplicaciones que se creen dentro de nuestra empresa.

Uno de los principales inconvenientes se encuentra al realizar la implementación de código dentro del aplicativo. Si bien contiene una amplia variedad de librerías y código reutilizable con tareas comunes, que lo hace robusto y con muchas posibilidades, el probarlo fuera del aplicativo se hace complicado. Es necesario contar con un entorno de desarrollo con plugins específicos para DataPower, pero aún así el código se suele ejecutar en remoto dentro del aplicativo mediante una llamada al mismo, nunca desde el entorno de desarrollo local. Esto finalmente es una limitación e implica una mayor complejidad a la hora de utilizar funciones propias de DataPower, lo que aumenta el coste final de los proyectos al aumentar su tiempo de desarrollo. Sí que se ha visto la ventaja de poder crear desarrollos extra a los que se han llamado simuladores para probar las comunicaciones seguras y todo lo relacionado con certificado y firma. Este tipo de prácticas no se describen claramente en ningún manual pero son muy interesantes y en este proyecto quedan bien definidas para su comprensión.

Otro gran problema que se ha solucionado es el relacionado con la gestión de los logs o trazas. El sistema en particular no permitía gestionar de manera ágil las mismas, además de que la revisión desde dentro del interfaz web no es útil. Finalmente y tras varias pruebas infructuosas, se optó por generar información en base de datos acerca de las transacciones y por enviar a un servidor externo las trazas de sistema. Por lo tanto, el disponer de las trazas en un aplicativo externo y en base de datos es una ventaja añadida la cual tiene mucho valor como implementación por todo lo que aporta: independencia por tener los logs en servidores externos y posibilidad de realizar auditoría de los procesos a posteriori.

Dejando de lado el punto de vista de sofware, el hardware escogido también juega un papel importante. Desde el tipo de modelo escogido, el DataPower XI52, hasta que sea en modo virtualizado. Respecto a esto último es necesario aclarar ciertos puntos. Para el desarrollo de este proyecto se ha escogido desplegar DataPower sobre una máquina virtual. La alternativa, el recurrir a una máquina física, resulta inalcanzable para un proyecto de este calibre. Sin embargo, se ha podido observar ventajas tales como la escalabilidad: la arquitectura propuesta se basaba en dos máquinas virtuales, una para las aplicaciones y trazabilidad, y la otra el propio DataPower. Por lo tanto es suficiente contar con una única máquina física para montar y levantar ambas imágenes virtualizadas. Es una ventaja en cuanto a reducción de costes, porque sobre un mismo servidor se pueden cargar ambas imágenes virtualizadas, pero una desventaja en cuanto a rendimiento y sobre todo seguridad. Siempre y cuando se las circunstancias no lo impidan, por ejemplo por una falta de espacio en los centros de procesado de datos, siempre se recomienda tener una máquina física y no virtualizada. Aunque en las pruebas de este proyecto no se pudiera concluir que el rendimiento del sistema se viera comprometido, si que es posible que en un entorno productivo, con una carga muy superior, se llegue a alcanzar este límite. Además, la seguridad extra que proporciona el dispositivo en formato *hardware*, como por ejemplo ante aperturas de la caja del mismo, es un factor que definitivamente inclina la balanza a favor de ese formato. Así pues, se va a recomendar adquirir el formato físico casi en cualquier cirscuntancia.

5.2 Líneas de trabajo futuras

Trabajar con una herramienta tan completa como DataPower ha proporcionado vías alternativas para la generación de cada una de las aplicaciones que se han ido desarrollando en el proyecto. Partiendo de la premisa de que lo mostrado puede considerarse como base

para el desarrollo de aplicaciones más complejas, no nos centraremos simplemente en describir las otras alternativas, sino que se presentan a continuación las líneas de trabajo a evolucionar a partir de los puntos clave terminados.

Se ha procurado afianzar una base en lo que respecta a la arquitectura genérica que acompañe a la máquina de DataPower. La configuración presentada a lo largo del proyecto nos permite gestionar más fácilmente la trazabilidad de las peticiones y respuestas. Una línea de trabajo a seguir sería el avanzar el método de trazabilidad para que permitiera guardar la estructura de datos completa y crear un mantenimiento en dichos sistemas. Los datos guardados incrementarían el espacio utilizando, teniendo que diseñar una arquitectura más avanzada para el almacenaje, ya sea en base de datos o en fichero.

Si bien es cierto que la arquitectura que se presenta es lo bastante genérica como para poder reutilizarse, sí existe una debilidad en la misma, y es el la ausencia de redundancia en los servidores que se utilizan. Al no disponer de dicha redundancia, un problema fatal en cualquiera de ellos puede provocar una indisponibilidad inmediata, generando pérdidas para la empresa. Por lo tanto, como complemento a dicho proyecto se puede avanzar en la mejora de la arquitectura para prevenir fallos de servicio.

Un detalle que a lo largo del proyecto ha estado siempre en el aire ha sido el de habilitar algún tipo de alarma en DataPower cuando existan problemas en los servicios web. No dispone de herramientas como tal dentro del mismo, pero sí que es posible utilizar la máquina de apoyo para tal fin dada la arquitectura propuesta. Ya que se está almacenando el detalle de las transacciones, realizando una extracción de los datos recopilados se puede crear una aplicación que los interprete y que ejecute algún tipo de alarma. Este punto es muy interesante ya que permite escoger entre una gran cantidad de tecnologías para la implementación de la aplicación con las ventajas que ello conlleva. Su implentación dotaría a la arquitectura de una monitorización proactiva realizada a medida.

Las alarmas anteriores, a nivel de lógica de negocio o interpretando los datos procesados, es una vía a desarrollar interesante. Pero no lo es menos el desarrollar otro tipo de monitorización del sistema a nivel de recursos de hardware. El aplicativo dispone de métodos para obtener información del mismo con el fin de tenerlo controlado en todo momento. Implementar un proceso que obtenga esa información, la analice y actúe en consecuencia es un camino a explorar dentro de la rama de monitorización y actuación proactiva.

Todo el análisis realizado ha sido expuesto siguiendo un protocolo concreto para las comunicaciones entre las empresas. Dentro del proyecto, en la definición del diseño, se expusieron las motivaciones y las bondades del protocolo de comunicación escogido para tal fin. A partir de esa elección, se realizaron los pertinentes desarrollos de cara a exponer un ejemplo práctico. La línea argumental a desarrollar consistiría en ampliar este ámbito de tres propuestas analizado en el proyecto mediante la aplicación de distintos protocolos ante las mismas situaciones. Así podría comprobarse las ventajas y desventajas de cada uno.

Por último, y la que se considera como la línea argumental más importante dada la proyección que últimamente existe, consistiría en la exposición de servicios REST mediante DataPower. El hilo de trabajo a desarrollar tomaría desde el análisis necesario para la exposición de una interfaz REST mediante DataPower, hasta la implantación en una red local y la transformación de Servicios Web a un entorno RESTful[93].

[93] RESTful se refiere a un entorno que cumple las características de la comunicación REST.

6 Manual de usuario e instalación

6.1 Aclaraciones

Este apartado contiene información sobre cómo se ha configurado la máquina virtual de DataPower en el PC personal para poder realizar los desarrollos y pruebas. En este caso, es una puesta a punto básica configurando aspectos tales como la configuración de red, los usuarios y los dominios. Para hacer un resumen de la configuración se va a mostrar la misma con comandos CLI[94]. Es útil saber que ejecutando esa secuencia de comandos sobre el intérprete de DataPower se puede obtener una copia idéntica de la configuración, por lo que la exportación de las mismas sirve para poder configurar otra máquina a modo de copia.

Además de la instalación de la máquina virtual de DataPower XI52, hay que realizar la instalación de software adicional en la máquina de apoyo para montar la arquitectura requerida:

1. Software de virtualización: "VMWare Player"
2. Base de Datos: "Oracle Database 11G R2 Express Edition"
3. Servidor de eventos syslog: "rsyslog"
4. Servidor de SFTP: "OpenSSH-Server"
5. Software para ejecución de pruebas: "SoapUI"
6. Editor de código XSLT: "Notepad++"

Este software se instalará sobre la máquina virtual de apoyo a DataPower, de modo que se tengan aisladas las configuraciones de red de las dos máquinas y se comporten como dos sistemas independientes. El sistema operativo elegido para la misma es Ubuntu 14.04LTS en su versión para 64 bits por tener un núcleo o *kernel* UNIX lo suficientemente estable y una amplia comunidad de usuarios apoyándolo, por lo que es más fácil encontrar documentación al respecto. Está claro que la elección final siempre depende de las necesidades imperantes, aunque se recomienda siempre una instalación basada en Linux por sus menores requisitos, por su estabilidad y por el bajo coste de mantenimiento.

6.2 Instalación de la máquina virtual DataPower XI52

Se parte de la última imagen de DataPower XI52 para VMWare disponible a fecha de Marzo de 2015: *xi7000_vmware_nonpd.ova* (firmware XI52.7.0.0.0). Su descarga se realiza desde la web habilitada por IBM a tal efecto a los *partners*.

Para arrancar la máquina virtual se utiliza "VMWare Player", ejecutándose en un PC sobre Windows 7, con CPU de 64 bits (opción "*Virtualization Technology VT-x*" activada en la *BIOS*) y con 8 GB de *RAM*.

Se opta por la instalación de DataPower con dos discos duros en modo redundante RAID[95]; normalmente la máquina física XI52 viene sin instalación de disco externo, pero se aprovecha la funcionalidad extra que nos proporciona la virtualización. Las ventajas de tener

[94] **CLI – *Command Line Interface.*** Es el lenguaje utilizado en la interfaz de comandos de DataPower. Es un lenguaje no basado en procedimientos y que contiene una serie de comandos orientados a tareas de configuración. Si bien es posible realizar todas las acciones del interfaz web mediante este lenguaje, es muy recomendable utilizarlo sólo para tareas básicas de configuración dada la complejidad que requiere. Es accesible mediante una conexión SSH; por defecto está deshabilitada por motivos de seguridad.

[95] **RAID:** *Redundant Array of Independent Disks*. Consta de un grupo de discos idénticos los cuales almacenan la información en modo espejo, duplicando los datos entre ambos y permitiendo recuperarla en caso de que alguno de los mismos quede inaccesible.

esta configuración es el tener redundancia en los discos para prevenir pérdida de datos en caso de fallo grave.

Para levantar la máquina virtual es necesario configurarla para utilizar 4GB de memoria RAM y una *CPU* de 4 núcleos sobre un sistema operativo sin identificar de 64 bits. La máquina se levanta con 4 interfaces de red Ethernet. Se configuran las dos primeras interfaces (*eth0* y *eth1*) en modo *Bridged* y habilitando el *check* para replicar la red a la que nos estamos conectando, de tal forma que se comporte como una máquina aislada en nuestra red interna, aunque realmente sea una máquina virtual sobre la misma tarjeta de red. El resto de interfaces se dejan desconectados puesto que no se van a usar, aunque es útil tenerlas en la máquina virtual en caso de necesidades futuras: conexiones a nuevas redes, pruebas, etc.

6.3 Configuración de la máquina virtual DataPower XI52

La configuración de la máquina virtual se realiza de manera similar a la de una física. Inicialmente se dispone de acceso vía terminal para realizar una primera configuración de la misma, ya que los puertos Ethernet están deshabilitados por defecto. Una vez hecha la configuración preliminar, se puede activar el acceso web vía HTTPs y SSH para poder configurarla desde otros medios (o lo que normalmente significa, configurar la máquina física en remoto y no en el CPD o Centro de Proceso de Datos en el que se encuentre).

Se habilita la conexión HTTPs y SSH para todas las interfaces de red conectadas, aunque lo más razonable es tener el tráfico de datos y de gestión separados, como se comentará más adelante. El acceso HTTPs proporciona un interfaz web visual, mientras que el acceso SSH proporciona un acceso mediante intérprete de comandos de DataPower. Los comandos CLI utilizados para activar ambas funcionalidades son los siguientes:

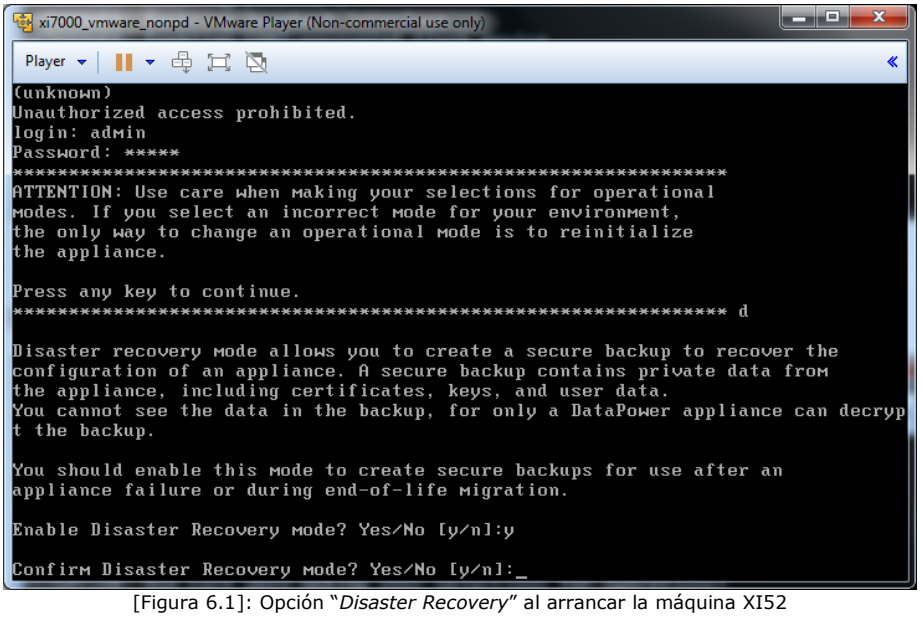
```
xi52(config)# web-mgmt 0.0.0.0 9090
xi52(config)# ssh 0.0.0.0 22
```

Un detalle muy importante a tener en cuenta al arrancar por primera vez la máquina es el activar la opción "*Disaster Recovery*". Con ella activada se permite exportar desde las opciones de DataPower, ya sea en modo Web o desde el intérprete de comandos CLI, la configuración completa de la máquina incluyendo las claves privadas. Suele ser muy útil en caso de migración a otra máquina ya sea por fin de soporte de la misma o por fallo irrecuperable que no permita arrancar, pudiendo intercambiar la estropeada por otra máquina nueva teniendo la misma configuración. Se ofrece la opción de habilitarlo solamente al primer arranque de la máquina por ser un posible agujero de seguridad, aunque los datos obtenidos de la exportación sólo sean descifrables por otro DataPower, por incluir información relativa a las claves privadas. Sin embargo, es muy recomendable activarlo puesto que la información relativa a las claves privadas no se puede extraer de la máquina de otra forma; la máquina se vende como una solución completa y cerrada, no es modular ni cuenta con posibilidad de ampliación y está físicamente sellado para evitar intercambiar módulos de memoria. Sí existe una posibilidad, pero es a costa de tener instalado un módulo hardware adicional que permita extraer y almacenar dicha información, el HSM – *Hardware Security Module*. No se suele adquirir en entornos productivos debido también a su aumento de precio respecto a la versión sin el módulo; aproximadamente 12.000 $ de diferencia[96].

[96] **HSM – Hardware Security Module**. Es un módulo que añade la funcionalidad de exportar la información relativa a las claves privadas de DataPower y que es totalmente opcional. Se compra conjuntamente con el aplicativo de DataPower. El precio del mismo se ha obtenido de la web oficial de IBM.

Por lo tanto, se recomienda habilitarlo y no ofrecer el usuario de administrador para usuarios genéricos, ya que es el único que va a poder realizar la exportación de todo el sistema completo.

A continuación se muestra una captura de pantalla en la que se muestra la consola de DataPower en el primer arranque realizado con la máquina virtual y en el que se configura la opción comentada anteriormente.



[Figura 6.1]: Opción "*Disaster Recovery*" al arrancar la máquina XI52

Como ya se ha explicado, la máquina DataPower XI52 utiliza una estructura basada en dominios. Cada uno de los cuales están accesibles por un determinado grupo de usuarios y pueden tener visibilidad de otro dominio existente. Cada dominio tiene su estructura interna de carpetas; además existen algunas carpetas compartidas entre entornos (como por ejemplo *sharedcert:///*). El dominio *Default* es el reservado para el administrador de la máquina. El resto de dominios son utilizados para usuarios con otros roles distintos del administrador, sobre los cuales se realizan los desarrollos de aplicaciones. Para este proyecto se añade un dominio adicional:

```
        xi52(config)# domain desarrollo
        xi52(config domain desarrollo)# admin-state enabled
        xi52(config domain desarrollo)# summary "Entorno de desarrollo PFC"
        xi52(config domain desarrollo)# visible-domain default  [up]
        xi52(config domain desarrollo)# file-permissions
CopyFrom+CopyTo+Delete+Display+Exec+Subdir
        xi52(config domain desarrollo)# file-monitoring
        xi52(config domain desarrollo)# config-mode local
        xi52(config domain desarrollo)# import-format ZIP
        xi52(config domain desarrollo)# local-ip-rewrite on
        xi52(config domain desarrollo)# maxchkpoints 3
```

Respecto a la configuración de red, se va a habilitar la interfaz *eth0* para sentido de comunicaciones hacia el exterior y la *eth1* hacia el interior; no se configura una interfaz de mantenimiento para el acceso de configuración, se accede por las de datos. Esto implica que el flujo de paquetes de datos y el relativo al mantenimiento y configuración van a compartir conexión. Normalmente en entornos empresariales esta configuración se realiza utilizando otra interfaz exclusiva, de nombre *mgt0*, para canalizar el tráfico de gestión. Con esto se

tendría un mayor control sobre los paquetes transmitidos y recibidos, sobre todo para monitorizar el tráfico de los mismos, y una mayor seguridad al poder implementar una red de confianza dedicada para acceder (distinta subred de la red de datos por ejemplo).

Este es el resumen de la configuración de la interfaz *eth0* configurada con comandos *CLI*. La interfaz *eth1* se configura de manera similar:

```
xi52(config)# interface eth0
xi52(config ethernet eth0)# admin-state enabled
xi52(config ethernet eth0)# summary "Conexion red local"
xi52(config ethernet eth0)# ip-config-mode static
xi52(config ethernet eth0)# ip-address 192.168.2.114/24
xi52(config ethernet eth0)# ipv4-default-gateway 192.168.2.1
xi52(config ethernet eth0)# standby-enable off
xi52(config ethernet eth0)# standby-group 1
xi52(config ethernet eth0)# standby-priority 100
xi52(config ethernet eth0)# mac-address 00:0c:29:19:18:e1
xi52(config ethernet eth0)# mode Auto
xi52(config ethernet eth0)# hardware-offload on
xi52(config ethernet eth0)# flow-control auto
```

Como se puede observar en los comandos CLI anteriores, se deja el modo *standby* deshabilitado. Este modo se utiliza en el caso de tener varias máquinas DataPower XI52 en tolerancia de fallos. Al activarlo se puede crear una IP virtual de modo que el tráfico de entrada se redirija a la misma, y luego sean los DataPower con el mismo *standby-group* los que lo procesen dependiendo del peso o *standby-priority* que tengan en la interfaz. La ventaja de este modo es que es totalmente configurable sin necesitad de ninguna máquina de apoyo, únicamente contando con los dos DataPower. Para nuestra arquitectura no va a ser necesaria esta configuración, pero sí sería interesante de cara a un entorno productivo real, ya que proporciona redundancia frente a fallos.

7 Referencias

A continuación se presenta el listado de referencias utilizadas en el proyecto, entre las que se incluyen manuales técnicos especializados, bibliografía y recursos Web. En caso de que exista posibilidad de descarga del recurso, se indicará su enlace. Se van a ordenar por aparición en el proyecto mediante un número secuencial entre corchetes *"[xx]"*. Para todas las referencias en línea se ha verificado su disponibilidad a fecha de cierre de este proyecto, Octubre de 2015.

[01] Definición de Web Services. (2002). http://www.w3.org/DesignIssues/WebServices.html

[02] Interacting with the SOA-Based Internet of Things. (2010). http://doi.ieeecomputersociety.org/10.1109/TSC.2010.3

[03] Glosario de términos de los Web Services. (2004). http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/

[04] Página del producto IBM DataPower Gateway. http://www-03.ibm.com/software/products/en/datapower-gateway

[05] TANENBAUM, Andrew S. *Redes de computadoras*. Pearson Educación, 2003. ISBN: 970-26-0162-2

[06] Sitio web de World Wide Consortium (W3C). http://www.w3.org

[07] Sitio web de OASIS. https://www.oasis-open.org/

[08] Definición del estándar WS-I Basic Profile. (2004). http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html

[09] Definición del estándar WS-I Basic Security Profile. (2007). http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html

[10] Request For Comments de la IETF. http://www.ietf.org/rfc.html

[11] Sitio web de The Internet Engineering Task Force - IETF. https://www.ietf.org/

[12] Hypertext Transfer Protocol HTTP. (1999). http://tools.ietf.org/html/rfc2616

[13] HTTP Over TLS. (2000). http://tools.ietf.org/html/rfc2818

[14] The Secure Sockets Layer SSL. (2011). http://tools.ietf.org/html/rfc6101

[15] The TLS Protocol. (1999). IETF: http://www.ietf.org/rfc/rfc2246.txt

[16] Definición de SOAP 1.1. (2000): http://www.w3.org/TR/2000/NOTE-SOAP-20000508

[17] Fielding, D. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[18] Extensible Markup Language XML. (2013). http://www.w3.org/XML/

[19] XML Schema. (2001). http://www.w3.org/XML/Schema

[20] Web Services Description Language WSDL 1.1. (2001). http://www.w3.org/TR/wsdl

[21] XML Entity Expansion (2010). The Web Application Security Consortium: http://projects.webappsec.org/w/page/13247002/XML%20Entity%20Expansion

[22] Sitio web de Oracle SOA Suite. http://www.oracle.com/technetwork/middleware/soasuite/overview/index.html

[23] IBM WebSphere DataPower B2B Appliance. (2009). IBM RedBooks: http://www.redbooks.ibm.com/redbooks/pdfs/sg247745.pdf

[24] Sitio web de Forum Sentry API Security Gateway. http://www.forumsys.com/products/forum-sentry-api-gateway/

[25] Sitio web de CA API Gateway. http://www.ca.com/us/securecenter/ca-api-gateway.aspx

[26] Sitio web de Axway API Gateway. https://www.axway.com/en/enterprise-solutions/api-gateway

[27] Sitio web de la consultora Lustratus Research Limited. http://www.lustratus.com/

[28] IBM presenta su candidatura a dominar el mercado SOA. (2007). Techweek: http://www.techweek.es/soa/noticias/1001604005601/ibm-candidatura-dominar-mercado-soa.1.html

[29] Foro de IBM para DataPower. https://www.ibm.com/developerworks/community/forums/html/forum?id=11111111-0000-0000-0000-000000001198

[30] Axway API Gateway in IBM DataPower Environments. https://www.axway.com/pt-br/enterprise-solutions/api-management/legacy-gateway-replacement/ibm-datapower#tablist1-tab1

[31] Lightweight Directory Access Protocol LDAP. (2006). http://tools.ietf.org/html/rfc4510

[32] Lightweight Third Party Authentication LTPA. (2014). http://www-01.ibm.com/support/knowledgecenter/SSCGGQ_1.1.0/com.ibm.ism.doc/Security/se00013_.html

[33] Documentación de WebSphere MQ V8.0. (2014). http://www-01.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.helphome.v80.doc/WelcomePagev8r0.htm?lang=es

[34] Documentación de la suite de productos de Tibco. https://docs.tibco.com/

[35] RFC de NFS. (1995). http://tools.ietf.org/html/rfc1813

[36] XSL Transformations XSLT. (1999). http://www.w3.org/TR/xslt

[37] Sitio web de Log Management - Syslog. http://www.syslog.org/

[38] IBM Tivoli Monitoring. (2013). http://www-01.ibm.com/support/knowledgecenter/SSDKXQ_6.3.1/com.ibm.itm.doc_6.3fp2/itm63fp2_qsg_en.htm

[39] Sitio web de JavaScript Object Notation JSON. http://json.org/

[40] OASIS Web Services Security. (2006). https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

[41] Manual HTML de JSONx. http://www-01.ibm.com/support/knowledgecenter/SS9H2Y_6.0.0/com.ibm.dp.xi.doc/json_jsonx.html

[42] Sitio web de Camerfirma. http://www.camerfirma.com/

[43] Sitio web de Thawte. https://www.thawte.com/

[44] Sitio web del producto IBM DB2 database software. http://www-01.ibm.com/software/data/db2/

[45] Oracle Database 11g Express Edition. http://www.oracle.com/technetwork/database/database-technologies/express-edition/overview/index.html

[46] HINES, Bill, et al. *IBM WebSphere DataPower SOA Appliance Handbook*. Pearson Education, 2008. ISBN: 978-0-13-714819-6

[47] Directorio público de Servicios Web SOAP. http://www.service-repository.com/

[48] WSDL del servicio utilizado para una comunicación extremo a extremo. http://www.predic8.com:8080/base/IDService?wsdl

[49] Simulador web de WS públicos. http://www.service-repository.com/client/start

[50] Plantilla para WSDL v. 2.0. (2007). W3C: http://www.w3.org/2007/06/wsdl/wsdl20.xsd

[51] Rodriguez, JUAN R., et al*. IBM WebSphere DataPower SOA Appliances Part III: XML Security Guide.* IBM RedPaper, 2008. Disponible en: http://www.redbooks.ibm.com/redpapers/pdfs/redp4365.pdf

[52] Rodriguez, JUAN R., et al. *IBM WebSphere DataPower SOA Appliances Part IV: Management and Governance*. IBM RedPaper, 2008. Disponible en: http://www.redbooks.ibm.com/redpapers/pdfs/redp4366.pdf

[53] Sitio web de IBM RedBooks. http://www.redbooks.ibm.com/

[54] Sung-Ik Son. *DataPower Problem Determination Techniques.* IBM RedPaper, 2008. Disponible en: http://www.redbooks.ibm.com/redpapers/pdfs/redp4445.pdf

[55] Sitio web de Oracle. http://www.oracle.com/es/index.html